D a t a F l e x

Version 2.0

Application Development Software System

DATA ACCESS CORPORATION

8525 SW 129 Terrace
Miami, Florida 33156
USA

Manual Revision Date: 07/18/84

MBASIC	Microsoft
CBASIC	Digital Research
CP/M	Digital Research
MP/M	Digital Research
WordStar	MicroPro International
MailMerge	MicroPro International
Pascal/MT+	Digital Research
dBASE II	Ashton-Tate, Inc.
PC-DOS	International Business Machines
IBM	International Business Machines
MS-DOS	Microsoft

A COMPLETE, UPDATED MANUAL WILL BE YOURS . . .

If we use a correction or improvement to this manual which you submit to us.

We're continually improving our manual to make the power of DataFlex easier to get to for the new user. User feedback is the chief source of corrections and improvements to the manual. The Disclaimer on the preceding page says that we aren't obligated to provide you with updates to DataFlex or its manual, but if we use a correction or improvement you send us, we'll send you a copy of the edition of the manual which contains your update(s), subject to the following: (a) If we receive the same suggestion from several sources, only the **first** person submitting the correction/improvement gets a new manual; and (b) If we use two or more of your suggestions in one new edition of our manual, you still only get one manual (multiple suggestions still increase your chances of being first on at least one of the suggestions). Data Access Corporation's decision in these matters is final. A new manual does not include a new binder (the old one is re-usable).

Here's How:

Just fill out the form on the opposite side of this page indicating what your suggestion is. If you have multiple suggestions, please make copies of the form. Please make your name, address, and phone clear so that we can get back to you if you win a new manual. Send the suggestion(s) to:

DOCUMENTATION
Data Access Corporation
8525 SW 129 Terrace
Miami, Florida 33156

...and we'll get back to you if yours is a winner! Either way, thanks for helping us help others to get more of the power of DataFlex.

SUGGESTION
For Improvement of the DataFlex Version 2.0 User's Manual

From: Submitted:

Name____ Date______________

Address______________________________ Phone (A/C)___ ______________

Address____

City ____________________________ State/Prov.________ Zip__________

Country______________________ Page Height:____in. x Width:____inches

Revision Date of Manual:____________ Page Number(s):_______________

TYPE OF SUGGESTION:

___ Correction of wrong information
___ Addition of missing information
___ Clarification/Improvement of existing example
___ Addition of new example

DETAILS:

____

____

____

____

____

____

____

I submit the above suggestion to Data Access Corporation with full permission to make any use of it may wish to, with the understanding that compensation to me for it, if any, will be completely at the discretion of Data Access Corporation. I assume no liability for its completeness or accuracy.

______________________________________ ___________________________

Signature DataFlex Serial Number

TABLE OF CONTENTS

SECTION C (continued)

SECTION D (continued)

SECTION D (continued)

LIST OF SAMPLE SCREENS

LIST OF SAMPLE CONFIGURATIONS

ORGANIZATION OF THIS MANUAL

SEQUENCE OF PRESENTATION

The sequence in which material is presented in this manual is the recommended sequence for the undertaking of tasks in learning DataFlex and in using it. Your learning experience with DataFlex will rapidly become a productive experience, in that within a couple of hours, you will be able to create database structures and control them with configurations. Configurations is the word used in this manual for a data processing procedure, or program, written in DataFlex. Your early exercises will yield simple, straightforward applications which despite their simplicity will have more than enough power to efficiently perform any number of common data processing tasks, such as mailing lists.

But you very likely acquired DataFlex because its power and flexibility go far beyond mere list management. As you continue reading this manual, and testing the facilities described in each section, possibilities for greater scope and sophistication will multiply rapidly until you reach the point at which you have the means of satisfying all of your data management needs. The simpler facilities of DataFlex are so powerful that you may reach this point well before you explore the more advanced features of DataFlex described in later sections. If this happens, there is no need to study further except to generally acquaint yourself with the capabilities of the more advanced features.

NOTATIONAL CONVENTIONS IN THIS MANUAL

Certain notational conventions are used throughout this manual with which you should become acquainted in order to understand the formats and examples shown. DataFlex configurations are made up of screen images (always at the beginning) and then command lines, the two sections being separated by the distinctive "/*" character combination by itself on one line. DataFlex commands are like sentences made up of a verb (the command) and (usually) one or more objects (referred to as arguments), sometimes followed by one or more options, and sometimes further followed by a comment. Commands may be preceded by conditional terms called indicators. Lastly, some command lines begin with "labels", distinguished by a terminating colon (LABEL:), whose purpose it is to "mark" a spot in a configuration for purposes of future references (such as a RETURN or GOTO command). Here is the typical format of a command line:

```
        [indicator] command argument {options} //comment
```

The brackets [] surrounding the indicator are required to identify indicators. The braces {} surrounding the options are not to be typed in most commands, except for format options for the ENTRY and FORMAT commands, where they are required. The braces are used in format illustrations, however, to signify an optional command element. Again, they are not to be typed in the command except in the cases noted. The "//" twin diagonals which precede the comment are, like the indicator brackets, part of DataFlex's syntax, and tell DataFlex not to compile the material following the diagonals, which leaves them free for any material you may wish to include in the source version of your configuration for reading by persons (as distinguished from the computer). Twin diagonals at the beginning of a line will cause the entire line to be a comment. Here is a typical command line:

```
        [FOUND] ENTRY mail.name {CAPSLOCK} // first try
```

This line starts with a (DataFlex-supplied) indicator, FOUND, which is enclosed in the required brackets, followed by the command, ENTRY, then the argument, which is a screen image window named (in the configuration) MAIL.NAME, followed by a formatting option (this one does require the braces), CAPSLOCK. It means, "If the desired record is successfully FOUND in the database, put the cursor into the next screen image window and accept for ENTRY into the database element MAIL.NAME the characters keyed into the window, but as though the CAPSLOCK key were on (lower case characters converted to uppercase)." The comment (first try) is not interpreted by DataFlex and hence has no meaning for execution; it would have meaning only for a person reading the configuration.

The example above illustrates the use of upper and lower case in examples throughout this manual. FOUND, ENTRY, and CAPSLOCK are all in capital letters, while mail.name is in lower case. This is intended to signify that FOUND, ENTRY, and CAPSLOCK are literal strings which match, respectively, an indicator, a command and an option which are inherently part of DataFlex. In any configuration which you wrote using these elements, you would have to literally use these strings then, also. Mail.name, on the other hand, is peculiar to the example or configuration which is being discussed, and would typically be some other string in any configuration written by you (although you could use it if you chose to). You will find this use of upper and lower case throughout the formats and examples in this manual, except for the full sample configurations, which follow the normal configuration convention of being written in solid capital letters except for textual material in screen prompts and messages which need upper and lower case.

Throughout DataFlex, in the system as well as in this manual, material enclosed in either single (') or double (") quotation marks is **literal** material which passes through configurations as such and is displayed or output, rather than executed.

DataFlex itself has many syntactical conventions in addition to the ones described above, which are peculiar to presentations in this manual. DataFlex's own syntactical conventions are described separately at the appropriate places in this manual.

THIS PAGE INTENTIONALLY LEFT BLANK

OPERATIONAL ORIENTATION

This section describes the steps involved in developing an application in DataFlex and the various files and programs that come into play both in creating and in running it.

DataFlex offers a wide range of facilities for applications development. Because of this, there is frequently more than one way to achieve a particular goal. Since one or another of the alternative methods is normally preferable under certain circumstances, care should be taken to avoid thinking that the first way conceived or implemented for doing a task is the only, or best, way. Any method developed for doing a task should be reviewed against the possibility that an alternative approach might be easier or more efficient.

This section is divided into two parts: Runtime System and Application Development. The runtime system is the part of DataFlex required to execute (run) the application. The person(s) who run the application is(are) referred to as the "operator" (as distinguished from the configurator, who is the person who develops the application). The application development section describes the parts of DataFlex which support development of applications. Although the configurator and the operator may in some instances be the same person, the activities of configuring on the one hand and operating on the other are different, so that it is useful to distinguish between these functions.

INSTALLATION

The installation of DataFlex consists mainly of copying the necessary files from your DataFlex master disks onto your working disks or hard disk drive and then using the DataFlex SETSCREEn program to initialize the system for your serial number and terminal type. The SETSCREEn program reads the file TERMLIST.CFG and writes the terminal information input by the user to the file FILELIST.CFG to be used by the runtime system and the configuration system. See the installation instructions and the section on Installing DataFlex for more information on installation.

RUNTIME SYSTEM

The basic operating nucleus of the DataFlex runtime system is the program "FLEX.COM" (FLEX.CMD on CP/M-86 systems). FLEX.COM and its overlay file "RUN.OVF" are functional all the time any DataFlex configuration is running. The DataFlex configuration contains the intelligence which enables DataFlex to make the computer do a particular task, just as a BASIC program works through the BASIC interpreter. Without a compiled configuration (.FLX) file, the runtime system can do nothing.

Under control of a configuration, the DataFlex runtime system can access and manipulate one or more databases, interact with the operator, drive the printer, read and write standard ASCII files, and otherwise activate just about every function found in microcomputer systems. These activities are controlled by configurations (files with the ".FLX" extension) created and compiled by the developer. When FLEX is run from the operating system, the name of the configuration which is to be run must be given with it.

When initialized, the DataFlex runtime system first reads a file called FILELIST.CFG which contains information about what database files are available, and information which enables DataFlex to generate the control codes which will operate your particular terminal.

Each data base file name appearing in FILELIST.CFG actually refers to a "family" of related files on your disk, all having a common root-name, but different extensions. The data files (".DAT" extension) contain the raw data in the sequence in which it was entered, and the basic information on the structure of the data file. The key files (".Kx" extension where x is the number of the index) contain index data which provides access to the data by indexes established by the configurator during the process of defining the database. The tag file (".TAG" extension) contains the names of the fields which make up the database.

Thus the minimum file complement required to run DataFlex consists of: FLEX.COM, RUN.OVF, FILELIST.CFG, at least one .FLX (configuration) file previously created by the configurator, and all database files (.DAT, .Kx and .TAG) for any database(s) declared in (used by) the configuration(s). To use the QUERY facility the QUERY.COM and QUERY.OVF program files must also be present.

The DataFlex Menu System that greets the user when DataFlex is initialized is itself a DataFlex configuration. The name of the menu configuration is MENU.FLX. The menu facility keeps the menus in a standard DataFlex data base file whose name is MENU.DAT. MENU.DAT is installed as file 49 in the list of active databases in FILELIST.CFG as supplied. Thus to use the menu facility you must have MENU.FLX and MENU.DAT on your system.

Other supplied programs may be required occasionally at run time. REINDEX reads a data file and rebuilds the keys (.Kx files) in case they are damaged by a power, media or system failure. Reindex is also used to maintain BATCH indexes and create "ad hoc" indexes. The FREL ((Fre)e (L)ist) program reads through a data file and verifies (and rebuilds if necessary) the list of deleted records in a data base file, and also deletes or extracts any records that appear damaged (typically as a consequence of a power, media or system failure). Both of these programs may be kept off-line until needed.

APPLICATION DEVELOPMENT

The process of developing specific applications in DataFlex involves creating configurations (files with the .FLX extension) and database files to store and manipulate data in the desired fashion. These files are created by use of a group of utility programs supplied with DataFlex. The .FLX files are created by running the DataFlex COMPILER program on "source" files. Source files can be created by any of several different means, including the EDITOR.COM program supplied with DataFlex, or any text editor or word processor which is capable to producing plain ASCII text files you might happen to favor. The source files contain all the various commands in DataFlex which cause the system to carry out the desired processes.

You will probably produce your first configuration using the AUTODEF utility. AUTODEF also creates the definition for a new database, and the data and key files necessary for its data. To use it, the first step is to "draw" an image of the input and query screen that you will use on your new database (see the section on "Formatting With Images"). The DataFlex EDITOR or your favorite text editor will serve for this purpose. AUTODEF reads this file and produces both a file definition (.FD) file and a configuration SOURCE code file (with a .FRM extension), ready to be compiled. The configuration produced by AUTODEF can perform full file maintenance on the database file it creates.

The function of the EDITOR program is to create, display and modify DataFlex configuration SOURCE and IMAGE files. The source files can be created from scratch with the EDITOR, or can come from one of the auto-definition utilities like AUTODEF, QUERY or READ. The source files produced with EDITOR are standard ASCII files, so most word processors or editors can be used instead of the DataFlex EDITOR if you are more comfortable with one of them.

The DataFlex QUERY program provides the user easy access to the information stored in any database existing on the disk drive. The QUERY facility can be used as an ad hoc report generator that requires little training to use. QUERY is entirely prompt-driven, making it quite practical for use by operators, but once a query has been successfully performed, it permits storing the query criteria in a file as source code for a report configuration. The source file can then be compiled to become a permanent report. QUERY requires all of the standard runtime database files to be present.

The DataFlex READ configuration generates a source program to import standard ASCII data files. READ is primarily intended for those who are converting to DataFlex from other systems.

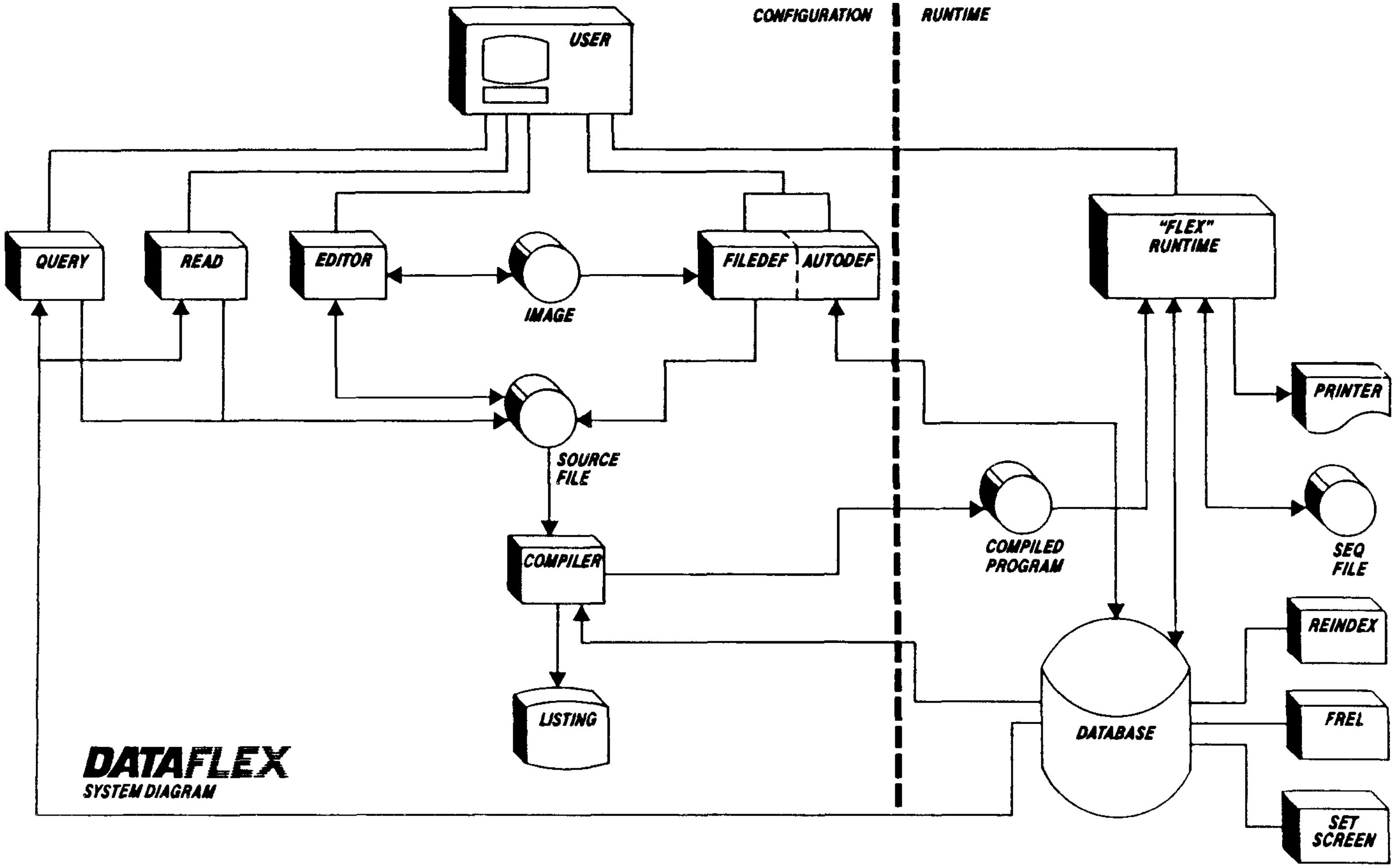
The CVTFRM and CVTRPT programs read DataFlex configurations written under earlier versions of DataFlex (1.6x) and produce DataFlex 2.00 source configurations. Configurations written in Versions 1.6x of DataFlex are fully convertible to configurations which run under Version 2.0x by use of these utilities. Naturally, configurations produced in this fashion will not take advantage of features of Version 2.0x which were not present in the version of DataFlex under which the original configuration was produced, but the converted source file can be edited to do so if desired.

The compiler (COMP.COM or COMP.CMD, and--for 8-bit systems--COMP.OVF) which produces .FLX files from source files requires three files to be present on the logged-in disk drive in order to do its job: the configuration source file, a file definition (.FD extension) file for each database declared by (used in) the configuration, and FLEX.CFL, the DataFlex command macro file. The .FD files and FLEX.CFL must both reside on the logged-in (default) disk drive. A successful compilation will produce a runnable DataFlex configuration with the .FLX extension and the same rootname as the source file. As the compiler runs, it will display on the screen the original source commands together with line numbers assigned to them by the compiler and any error messages which arise during compile time. This output can be directed to any of three devices: the console, the printer, or a printable (.PRN) file.

A DataFlex configuration that does not use any database(s) would not require any .FD files in order to be compiled, but the typical DataFlex configuration will access one or more databases. The .FD files are used only at compile time; they are not required at run time. On the other hand, the other runtime database files (.DAT, .Kx, .TAG) are not required by compile. This difference in file requirements can be taken advantage of when operating within restricted disk space.

The FILEDEF utility, like AUTODEF, is used for creating database files and configuration source files. FILEDEF is more versatile than AUTODEF. Multi-file relationships can be created with FILEDEF. FILEDEF requires more extensive knowledge of DataFlex, but gives more control over the database. The FILEDEF program can also read and write database definition (.DEF) files in standard ASCII. Definition files are useful for reference, database documentation and transfer of applications to other environments.

The PACK program is for use by users who want to create their own new commands and add them to the DataFlex language. The PACK program reads the file FMAC and creates the file FLEX.CFL which is used by the compiler. New command macros are added to FMAC with a text editor or word processor and then made usable by running PACK on the new version of FMAC, creating a new version of FLEX.CFL (overwriting the old one).



DATAFLEX
SYSTEM DIAGRAM

After running SETSCREEn and "informing" DataFlex of the codes which operate your terminal, there are three basic levels of using DataFlex through which you should proceed in order for the easiest, most thorough learning experience. The content of this manual is itself presented in the order which will best facilitate this approach. They are described below:

QUERYING AND REPORTING Your DataFlex system was supplied with more than a dozen working configurations complete with sample data, all on disk, compiled and ready to run. They are intended as learning aids and are not documented in this manual. You will find most of them listed in the "DataFlex demo files" submenu, which you may choose from the master menu. From this submenu, you may actually run any of the configurations, and/or query their databases, by electing the last choice, "QUERY Data Files." You should tour these samples freely, entering additional test data where you choose, and running reports to the screen and/or your printer from them at will. The Operator's Guide in this manual should provide all the guidance you will need in this activity. Eventually, you may even wish to modify the configurations themselves to make them run differently. (Some of the configurations may happen to be already very close to something you could actually use for your own purposes.) The source "code" (which is what must be modified and recompiled to change the configuration) is supplied for every configuration on the C: disk.

PROMPTED DATABASE CREATION AND CONFIGURATION DataFlex provides numerous "automated" methods for creating and maintaining databases and for creating the configurations you will need for entering data to, and reporting data from those databases. While the "shortcut" methods are prompted, and typically more narrowly structured than the "programming" methods of writing DataFlex configurations, they should not be underestimated. In actuality, they are highly flexible and very powerful in their own right. Roughly **two out of four** DataFlex users will have no regular occasion to use any DataFlex facility other than EDITOR (for writing configurations), MENUDEF (for creating and editing menus), AUTODEF (for creating databases and entry configurations) and QUERY (for creating reports and doing interactive inquiries). Of the remaining two out of four typical DataFlex users, **only one** will have requirements which cannot be fulfilled by the aforementioned facilities plus the ENTER macro and the REPORT macro. The facilities mentioned above should be explored thoroughly. You should proceed on to the next stage only in the event that you have unusual requirements, or a driving curiosity as to the virtually limitless possibilities offered by the most extensive, flexible applications development medium available in today's market.

PROGRAMMING AND EXTENSION OF THE LANGUAGE In line-by-line programming of applications, DataFlex offers facilities for the creation of applications not even yet imagined by its creators. The more fundamental functions provided in function key commands, sequential input/output, console input/output and multi-user functions allow the construction of virtually any processing structure which could be desired. Although whole configurations can be made up of such commands, these commands are more likely to be used intermingled with structures created by the pre-programmed facilities such as the REPORT and ENTER macros. While exclusive use of these facilities for writing configurations is likely to be a less efficient way of creating configurations, occasional use of them to alter the way standard structures function is very much encouraged. In the Advanced section of this manual, you will even find procedures for modifying DataFlex commands themselves, as well as procedures for creating entirely new commands for your individual use.

The schematic on the next page offers a graphic depiction of the ranges of programming sophistication to which each of the facilities of DataFlex primarily applies. The chart should not be interpreted too literally. There is no reason whatsoever why, for example, the "basic" facility of QUERY should not be used in an otherwise "advanced" application to create one or more reports if QUERY can satisfy the application's requirements for that particular function. Further, a common way of creating highly complex applications involves the use of "basic" or "intermediate" facilities such as QUERY or AUTODEF to create the "nucleus" of the application, and then subsequent editing of the nucleus to incorporate the more "advanced" features required by the application.

THE FACILITIES OF DATAFLEX

```
                     --------LEVELS OF CONFIGURATION SOPHISTICATION-------
   OPERATOR               BASIC          INTERMEDIATE        ADVANCED

¦¦¦¦EXISTING¦¦¦¦¦¦
¦¦CONFIGURATIONS¦¦

¦¦¦¦¦¦QUERY¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

¦¦¦¦¦MENU¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

¦¦¦¦AUTODEF¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                    ¦¦¦¦EDITOR¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                    ¦¦¦SCREEN IMAGES¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                    ¦¦¦¦COMPILER¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                    ¦¦¦¦MENUDEF¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                    ¦CONTROL COMMANDS¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                                      ¦most commands¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                                      ¦¦¦¦INDICATORS¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                                      ¦¦¦REPORT MACRO¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                                      ¦¦¦¦ENTER MACRO¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                                      ¦ENTERGROUP MACRO¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                                      ¦¦¦¦¦FILEDEF¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

                                                        ¦¦KEY PROCEDURES¦¦

                                                        ¦¦¦¦FIELDINDEX¦¦¦¦

                                                        ¦MACRO OPERATIONS¦
```

THIS PAGE INTENTIONALLY LEFT BLANK

OPERATOR'S GUIDE

The objective of this section is to provide a clear, non-technical summary of use of the system for data entry. The first part of the chapter will define the procedure to use DataFlex for data entry and file maintenance. Secondly, the features and facilities that may be used in any given application will be discussed.

DataFlex uses a set of command keys, called "Flex-Keys", to execute many of the regularly used functions in an application software system. Example Flex-Keys are FIND, used to locate a record in the data base; SAVE, used to store a new record in the data base, etc. Further, since DataFlex is used on such a wide variety of systems, especially terminals, we will address functions of the program by their Flex-Key name in the course of our discussion, rather than by the particular key on your terminal which has been established, since the location of that command key may vary from system to system.

The SETSCREEn Utility is used to assign the DataFlex Flex-Key commands to a set of control or function keys on your keyboard. Part of its operation is to print a list of the commands and the keys to which they are currently assigned on your terminal. If you do not have a copy of that list, go to the SETSCREEn Program now, and print one for use with this chapter.

FUNCTIONAL OVERVIEW

A DataFlex application that involves data entry has the capacity to accept data from the keyboard, check to make sure that it is the proper type of data, and then store it on the computer's floppy or hard disk storage device. Once data is stored, DataFlex can be made to retrieve (FIND) the data and display it on the CRT, and allow the operator to change or delete it.

All of the capabilities of DataFlex may or may not be used in every application. For example, the option exists within a configuration to lock an operator out of a particular data window on the screen or to automatically make calculations that are routine in the process of certain data entry situations. Our point is that although all of DataFlex's capabilities will be discussed here, you may not see all of them used in every application which you run on your computer.

DEFINITION OF TERMS

Control Key:	A key board entry that is made while the "control" key is held down. Variously represented as ^A or CTRL A to indicate that the control key is being held while the "A" is also pressed. Flex-Key functions can be assigned to control keys with SETSCREEn.
Data Entry:	The process of putting new information into the computer through the terminal keyboard.
File Maintenance:	The examination, changing, or deletion of information that is already stored in the computer.
Fill Character:	A character displayed in a "window" (see below) which defines the length of the data field which can be entered or displayed at that particular position. Generally, the underline character (_______) is used for this purpose.
Function Key:	A special key on the terminal keyboard that is outside of the normal typing array. It generates a coded signal that can be "trapped" by DataFlex to execute one of the DataFlex Flex-Key functions (e.g., FIND, DELETE, SAVE, etc.). Terminal function keys generate characters that do not display on the screen.
Index:	An organized list of keys which is linked to the records in a file. The index is maintained in order automatically by DataFlex so that records can be accessed sequentially even though the data is entered and stored randomly in the data base.
Key:	An element of data in a record by which the record can be accessed or "found" in the data base. E.g., the "ACCOUNT NUMBER" and/or "ACCOUNT NAME" could be defined as keys for an accounts receivable record; the "ITEM ID" and/or "DESCRIPTION" could be keys for an inventory record.
Key Field:	The field(s) in a data record which contains a key.
Key List:	Same as Index.
Window:	An area of the CRT display defined to accept or display data. Windows can be defined to accept only certain types of data... e.g. numbers only.

HOW TO BEGIN

Data entry and file maintenance are begun by selecting the proper option from the system menu. All that you need to do is select the number of the menu item for the data entry configuration that you wish to execute.

DataFlex screens are operated by the Flex-Key commands which you (or the operator) issue(s) from the keyboard. The commands are assigned to certain keys (e.g. TAB, LINE FEED, DELETE, RUB-OUT, etc.), function keys (F1, F2, etc.), and/or control sequences (^A, ^B, etc.) for your particular system. This assignment is done with the DataFlex configuration utility called SETSCREE. Once the assignment of the commands is made to the various keys on your terminal, all DataFlex screens will operate exactly the same regardless of the file(s) being processed or the format of the screen.

This feature of DataFlex promotes operator efficiency and reduces the opportunity for errors since there is a procedural continuity throughout the system. The execution of the commands soon becomes second nature so that when a new or altered module is added to the system, no additional training or familiarization is required.

DataFlex screen displays are "active", which means they can execute data entry or file maintenance functions at any time that a Flex-Key command is issued.

> For example, if a particular screen contains twenty display windows, but only the first five elements of data are to be entered initially, then a SAVE command can be issued after the fifth data item is entered. This makes it unnecessary to strike the RETURN key after each of the blank fields just to exit the screen. This feature can improve data entry efficiency by substantially reducing the number of keystrokes required.

FLEX-KEY SUMMARY

COMMAND	ACTION
ESCAPE	Terminate the ENTER Program, return to menu.
CLEAR	Clear all screen windows
RETURN	Terminate entry in the window where the cursor is currently located and move cursor to the first position of the next window.
BACK FIELD	Move the cursor from the current window back to the previous display window on the screen.
FIND	Find & display the record which is associated with the key in the window where the cursor is currently positioned. If there is no data in the window, find and display the record which is first key in the index for the field associated with the current window.
SUPERFIND	On a screen display which contains data from multiple files, this command will find a record by whichever key window the cursor is currently positioned in. Then, the related records to the keyed record will be found and displayed to fill all data windows on the CRT.
SAVE	Stores the displayed screen information to the data base. If the screen contains data from more than one file, data will be written to whatever files are part of the configuration.
DELETE	Removes the displayed record from the file and its key from the key list. A record must be found before it can be deleted.
PREVIOUS RECORD	Finds the record whose key immediately precedes the key in the window where the cursor is currently positioned.
NEXT RECORD	Find the record whose key immediately follows the key in the window where the cursor is currently positioned.
CALCULATE	Performs the calculation indicated by the expression entered. Calculations can be performed in a data window, as well as on the prompt line of the CRT.
HELP	Display HELP screen.
Window Edit	Move cursor or change data in an image window.

KEYBOARD COMMAND EXPLANATIONS

ESCAPE

When the ESCAPE command is issued from the keyboard, DataFlex immediately begins the process of "shutting down operations". All open data files are closed, and the control is returned to the menu. It is important to note that when the ESCAPE command is made, any data which is currently displayed is not acted upon in any way unless the program internally executes a set of commands. Specifically, if a new record has been entered, but not SAVEd to disk(ette), ESCAPE will not execute the SAVE command as part of its shut down. This also applies in the case of a changed or edited record: if a record is found and edited but not re-SAVEd, no change will be made in the data base.

CLEAR

The function of the CLEAR command is very simple... to erase any data from the display windows (only) and place the fill character in all data window positions. CLEAR has no effect on data stored in the data base whatsoever; it is solely a screen operation. CLEAR can be used as a "bailout" procedure during editing. If a record has been called to the screen (found), and an edit made that is not correct, CLEAR will erase the screen and preclude the bad edit from being written to the data base.

RETURN

The RETURN command terminates data entry in one display window and moves the cursor to the next window. This command is almost always assigned to the RETURN or ENTER key of your terminal.

If you are entering data, and the RETURN command is issued at the end of the last window on the screen, the displayed information will be automatically SAVEd in the data base.

If, during data entry, you wish to skip over a window and not enter any data, simply execute the RETURN command prior to entering any data. If the window accepts alphanumeric data the field will be given the value of a string of spaces that is the length of the window. If the window accepts numeric data, the field will be set to zero.

DataFlex can be made to do an "auto-return" at the end of a data field. This means that when entered data completely fills a window, the RETURN command will be executed automatically. If "auto-return" is "ON", the cursor will jump from the end of the current window to the beginning of the next window as soon as the current window is filled with data. The use of this feature is up to the user and configurator. It is turned "ON" with the SETSCREE Utility program.

FIND

A screen image will display a series of data windows where you can enter new information or recall existing information for display. Some of the display windows are "key fields" (as defined above). That is, they can be used to FIND records in the data base that are associated with data entered in the windows of the display. Generally, the first window in an image is defined as a key, however this is not always the case.

To FIND a particular record, place the cursor in the window for the key field (DESCRIPTION in the example below), enter the "value" of the key for the record you wish to FIND, and execute a FIND command. DataFlex will then retrieve the record with the value you entered in its key field and display it on the screen. If a key with the value which you entered does not exist, then DataFlex will FIND and display the first record whose key sequentially follows the position which the value you entered would occupy in the index.

To illustrate the way FIND operates, suppose we had a database with 5 records in it, each of which has two fields, DESCRIPTION (this is the key field), and VENDOR (which could also be a key field, but in this example, it will not be). This is the data:

DESCRIPTION	VENDOR
ARMATURE	General Electric
DISKETTE	3M Corporation
ESCALATOR	Otis Elevator
FRISBEE	Wham-O
KEYBOARD	Televideo

Here is what the following entries would find if they were entered into the DESCRIPTION window, and the FIND command executed:

	DataFlex would FIND	
YOUR ENTRY	DESCRIPTION	VENDOR
A	ARMATURE	General Electric
ARMATURE	ARMATURE	General Electric
B	DISKETTE	3M Corporation
C	DISKETTE	3M Corporation
DISK	DISKETTE	3M Corporation
E	ESCALATOR	Otis Elevator
F	FRISBEE	Wham-O
G	KEYBOARD	Televideo
H	KEYBOARD	Televideo
J	KEYBOARD	Televideo
K	KEYBOARD	Televideo
L	-Error - Find Past End of File	

SUPERFIND

On a screen that displays data from only one file, SUPERFIND is functionally identical to the use of the FIND command. However, on a screen which displays data from multiple files, SUPERFIND is a powerful tool to find and display information from multiple, related data base files.

The action of SUPERFIND is to use DataFlex's data base manager to retrieve all related data on a given screen. To use SUPERFIND, place the cursor in the window by which you wish to start the FIND. The window must be a key field. Enter the key value to find and execute the SUPERFIND command. If you want to start viewing the records from the beginning of the file in which the cursor is located, just execute SUPERFIND and DataFlex will automatically FIND the records related to the first key in the key list. Then, when the record is found, all related windows from other files will also be found and displayed. When SUPERFIND is used, the entire screen display is filled with data.

Once the screen display is complete, you can use the NEXT RECORD and PREVIOUS RECORD commands to sequence through the related data.

Consider the following example:

We have an inventory order entry system... the files are:

PARTS	Key = Part #
BUYERS	Key = Buyer ID
TRANSACTIONS	Key = Transaction #

Data entry configurations are used to enter all data into the system. PARTS and BUYERS can be added and maintained and items can be ordered through the creation of TRANSACTIONs. TRANSACTIONs contain the Buyer ID, the Part #, the quantity of the PART sold and a total sale amount.

The stage is set... if we now load the transaction screen, place the cursor over the Buyer ID window, enter a Buyer ID and press SUPERFIND, all fields will be filled. Starting with the entered Buyer, we can now issue the NEXT RECORD command and if the displayed buyer has more transactions, the next transaction will be found and displayed on the CRT. In other words we are now FINDing transactions by Buyer.

The fun has just begun... If we execute a CLEAR command, place the cursor in the PART # window and enter a PART, SUPERFIND will now FIND the PART and its related transactions. NEXT (or PREVIOUS) RECORD will sequence up and down the transaction list for that PART. When we hit the end of the transactions for the requested PART, the next PART in the index will be displayed with its transactions.

SAVE

The action of the SAVE command is to take the data displayed on the screen and save it to the correct data files in the data base. This can occur whether there is one or more files associated with the display. SAVE is a simple yet powerful command. If a configuration is not carefully conceived, SAVE can allow unwanted changes to the data base from a display containing data from several files.

There is no "edit" command in DataFlex. Editing is accomplished with the use of the FIND and SAVE commands. First the record to be edited is brought to the screen with the FIND command. Then the necessary edits or changes are made on the screen. When the data is correct, a SAVE command is executed to place the altered record back in the data base. As mentioned above, if the edit is incorrect or needs to be aborted for any reason, the CLEAR command will remove the "found" record from the screen without altering any data.

DELETE

The DELETE command removes a record from the data base. Procedurally, DELETE can only operate following the execution of the FIND command (i.e., you must enter the key of the record to be DELETEd, FIND it, then execute the DELETE command). The record itself and its key(s) will be removed from the files.

NEXT RECORD

This command allows you to sequence through a file, displaying records in key order. When the cursor is placed in a key window, the NEXT RECORD command will FIND the record whose key is next in the index for that window's data and display it. If multiple keys are used on the screen, the cursor can be located in any valid key window for the operation of the NEXT RECORD command.

PREVIOUS RECORD

Same as the NEXT RECORD command except that it FINDs the record whose key precedes that of the one currently displayed.

CALCULATE

The CALCULATE command allows the operator interactive access to the arithmetic capabilities of DataFlex. It can be used in two locations on the screen: in a window, or on the command line at the bottom of the CRT. The format for the calculation is the same regardless of the location where it takes place on the screen. Four symbols are used for arithmetic operation:

+	=	Addition
-	=	Subtraction
*	=	Multiplication
/	=	Division

The arithmetic expression can simply be typed in, e.g., 2+2, 10-7, 3*50, or 24/6. Parentheses are supported, so any of the following are valid expressions for use by the CALCULATE Command:

```
(2+2)*(5/6)   (2*(2*(2*(2)))   100/34+12*(365/12)
```

If the cursor is located in a window which contains numeric data, an arithmetic expression can be entered directly. When a RETURN command is executed, the arithmetic will be performed according to the expression that was entered. No command need be used in this case.

If the CALCULATE command is issued, the cursor will go to the bottom line of the display, where the expression can be entered. When the expression is complete, execute a RETURN command and the arithmetic will be performed with the result displayed in the window where the cursor was located when the CALCULATE command was issued. This applies whether the cursor was originally located in a numeric or alphanumeric window.

HELP

The HELP command will cause an application-dependent "help" screen to be displayed. The screens are created as part of an application and are covered in the section of this manual on "Formatting with Images". Syntax for using the HELP command within an application to call the HELP screen is described in the section on "Using Function Keys".

USER-DEFINED

An additional key on your terminal may have been assigned some additional function in your DataFlex application(s). If so, this was done in your particular program, and your supervisor or programmer will tell you which key it is, if any, and what it does, or you will find it in the manual for the particular application(s) you are running.

RIGHT & LEFT Arrow Window Editing Keys

These keys non-destructively move the CRT cursor within a DataFlex screen display window during data entry or editing.

BACKSPACE Window Editing Key

Moves the cursor one space to the left in a screen window, deleting the character under the cursor, as well as any characters which may be in the window to the right of the cursor.

CHARACTER INSERT Window Editing Key

Inserts a space at the current cursor position in a screen window. Characters to the right of the cursor will be moved one space to the right.

CHARACTER DELETE Window Editing Key

Deletes the character under the cursor in a DataFlex screen display window. Characters to the right of the cursor will be moved one space to the left.

INSTALLING DATAFLEX

The DataFlex SETSCREEn utility program allows you to configure Data-Flex to run on your CRT terminal, and to specifically adapt certain operations of the software to your specific requirements.

In this chapter of the User's Manual, we will use the word "terminal" to refer generally to the keyboard and video display of a system, whether the device is separate from the computer itself or, as in cases like the IBM Personal Computer, the keyboard and video display are directly connected and integrated with the computer.

SETSCREEn can be entered directly from the operating system command level by typing the first eight letters of the program name:

```
        A>SETSCREE      <RETURN>
```

Upon entry to SETSCREEn, a list of pre-defined terminal and system configurations will be displayed as in the example below. You have the option of selecting a predefined terminal which is on the list, adding the data for a terminal which is not on the list, or editing the data for a terminal which is already on the list.

```
 1   SUPERBRAIN                        2   ADM 31
 3   V.C. 404                          4   TELEVIDEO (950/925)
 5   HAZELTINE 15xx                    6   ADM 3A
 7   TRS-80 ( P&T CP/M )               8   MICROTERM ACT 5
 9   ADDS VIEWPOINT                   10   HP 125
11   IBM PC W/ PCTERM (dos 1.1)       12   DEC VT-52
13   XEROX 820                        14   ZENITH 100
15   NEC A.P.C.                       16   ANSI TERMINAL (DEC)
17   HEATH H19/ZENITH Z19             18   T. I. PEGASUS
19   IBM PC W/ ANSI.SYS

20   MY TERMINAL IS NOT ON THIS LIST
```

If your terminal appears on the list displayed, simply enter the number by the name of your terminal. This will select the proper screen codes and keys as preconfigured for your terminal and function keys that DataFlex will act upon. The predefined Flex-Key assignments provided for the terminals may be changed at any time by use of Option Two in the function menu shown below. If you have selected a displayed terminal, the following menu will be displayed:

```
  EDIT:
        (1) SET SYSTEM OPTIONS
        (2) ASSIGN FUNCTION KEYS
        (3) SET TERMINAL CONTROL CODES
        (4) DISPLAY PRESENT CONFIGURATION
        (5) PRINT CONFIGURATION
        (6) END / SAVE CHANGES
        (7) ABORT SETSCREEN - DO NOT WRITE CHANGES
```

You may wish to use Option 4 to display the configuration to check it over and/or print the configuration to paper for future off-line reference. If it is necessary for any reason to edit terminal control codes or function key assignments, these may be done by use of options 3 and 2 respectively. Option 6, (END / SAVE CHANGES) will write the configuration to disk for future automatic use by DataFlex.

NOTE: There is a "standard" for terminals that is being used more and more, called the ANSI terminal standard. If you cannot find terminal information in your manual, it may be compatible with the ANSI standard. Many of the new "integrated" systems use the ANSI standard. Systems known to use the ANSI standard are: Televideo 970, DEC Rainbow, Texas Instruments Professional, Wang PC, and IBM-PC with DOS 2.0. Choice number 16 above will provide the ANSI configuration.

If your terminal (or a terminal which your terminal emulates) is not listed, it will be necessary for you to get out your terminal or system manual and manually configure DataFlex using the "MY TERMINAL IS NOT ON THIS LIST" option. If you need to end SETSCREEn at this point, select any predefined terminal option number, and when the menu shown above is displayed, enter an option 7 to abort and return to the operating system.

If you have selected "MY TERMINAL IS NOT ON THIS LIST", and you have a terminal manual ready, it is time to proceed. The next prompt will ask for the name of your terminal. This will identify the entries that you make for future reference from the list of terminals initially displayed when SETSCREEN is loaded.

```
        ENTER YOUR TERMINAL NAME (UP TO 20 CHRS) :
```

From here, SETSCREEn will step through the first three options of the menu shown above with the following series of questions:

PART ONE: SYSTEM OPTIONS

The first system option is:

```
         DO YOU WANT AUTO RETURN ON FULL FIELD (Y OR N) N
```

This question is answered "Y" if, during data entry, you want DataFlex to move the cursor from one image "window" to the next automatically when the window has been completely filled with data. For example, if a window is formatted for date entries (8 characters including the slashes "__/__/__"), and you put in 1/1/83, the cursor would remain in the window until you pressed <RETURN>; however, if you entered 01/01/83, filling the window completely, the cursor would automatically move to the next window in the screen image.

Answer "N" to this question if you want the cursor to remain in a window until <RETURN> is pressed, at which time DataFlex will then (and only then) move the cursor to the next window.

The next system option is:

```
        IS THIS A MULTIUSER SYSTEM (Y OR N) N
```

If your DataFlex license is a multiuser version, you MUST respond with "Y" to this question to use your system in multiuser mode. On single **user systems always respond with "N".**

If you responded "Y" to the multiuser question, the next system option is (this question will not display for single-user systems):

```
        MUST THE DISK DATA BE PRE-ALLOCATED? (Y OR N)
```

If you answer this question "Y"es, a file will be created each time you establish a database (see Designing Databases) which is large enough to contain all the data your database can hold with the full number of records which you will later specify in establishing your databases. This question should be answered "N"o unless: (1) your operating system requires it (some multi-user systems do--if yours does, the fact is noted on the installation notes for your network); or (2) you require for some other reason for the maximum disk space your database could occupy to be pre-reserved. Pre-allocation of disk space can slightly improve performance in large databases with multiple files and numerous complex indexes.

When these three questions have been answered, you will be prompted:

```
         IS THIS DATA CORRECT (Y/N) Y
```

If you have made a mistake in an entry, a response of "N" will repeat the questions. If the entries are correct enter "Y" to proceed.

PART TWO: SET FUNCTION KEYS - FLEX-KEYS

In this section you will be asked to press certain keys on your keyboard which will be used throughout DataFlex to execute the Flex-Key functions (FIND a record, SAVE a record, etc.). DataFlex will read your entries from the keyboard as you make them and retain the internal codes generated as the proper response for the particular Flex-Key function.

The keys selected as Flex-Keys must not be ones that are used to generate characters during normal data entry (numbers, letters, periods, commas, etc.). You can use tab, line feed, delete, arrows, here-is, and any control character (formed by pressing the CONTROL key and a letter at the same time), or any "function key" (usually an extra row of numbered keys on the top row or on the left side of your keyboard).

If you have function keys, they do not have to be used to execute the Flex-Key commands; that is your option. SETSCREEn will figure out what kind of function keys your terminal has by asking you to press one (or two) of them. The function keys you press as Flex-Keys must all have the same "lead-in" code. Some terminals have "grouped" function keys with different lead-in codes, in which case you can only use one group. You will be prompted:

```
        DOES YOUR TERMINAL HAVE FUNCTION KEYS? (Y OR N) N
```

If you have, and want to use, function keys, answer "Y". Otherwise press <RETURN> to make a default entry of "N".

After you have established whether or not your system has function keys, you will be asked to press a series of keys to assign keys to a particular Flex-Key function.

```
        PRESS KEY CORRESPONDING TO FUNCTION
```

...will lead off the following list:

RETURN OR ENTER KEY	RETURN (13)
ESC (EXIT PROGRAM)	ESCAPE (27)
PREVIOUS FIELD (BACK FIELD)	LINE FEED (10)
DATA WINDOW CHR (E.G. "_")	"_" (95) (must be printable char.)
FIND A RECORD	TAB (9)
SUPERFIND	CTRL F (6)
SAVE A RECORD	CTRL S (19)
DELETE A RECORD	CTRL D (4)
PREVIOUS RECORD (SEQUENTIAL)	CTRL P (16)
NEXT RECORD (SEQUENTIAL)	CTRL N (14)
CALCULATE FUNCTION KEY	CTRL C (3)
CLEAR DATA WINDOWS ON SCREEN	CTRL A (1)
HELP KEY	CTRL Q (17)
USER DEFINED FUNCTION	CTRL U (21)
BACK SPACE(DESTRUCTIVE)	DEL/RUBOUT (127)
LEFT ARROW(NON DESTRUCTIVE)	CTRL H (8)
RIGHT ARROW	CTRL L (12)
UP ARROW	CTRL K (11)
DOWN ARROW	CTRL V (22)
INSERT CHARACTER	CTRL Z (26)
DELETE CHARACTER	CTRL X (24)

IS THIS DATA CORRECT (Y/N) Y

This list is the "standard" assignment of Flex-Keys that can be implemented on any system since it uses exclusively universal keys for each function. You may set up the Flex-Keys any way you want on your system. Then all DataFlex programs running on your system will use your assignments. Refer to the operator documentation for the meaning of each of these Flex-Key functions.

NOTE: You may not have duplicate assignments for a Flex-Key. Some terminals will generate the same code from two keys, typically back space and back arrow. If this is the case with your computer, you may NOT use both of these keys. The two forms of backspacing provided by DataFlex are: destructive (deletes data) and non-destructive (moves the cursor back without deleting data previously entered). They MUST be assigned to keys that generate different internal codes.

Except for the data window character, all of the above must be ASCII non-printable characters.

PART THREE: SET SPECIAL SCREEN FUNCTIONS

In this section you will configure DataFlex to clear the screen, set highlighting, and other screen attributes. All data in this section will be entered as decimal numbers with spaces separating entries that require multiple values (e.g. ESC "=" are two values that might be entered as 27 61). Up to four (4) values can be entered for each function. It will be necessary for you to turn to the screen control section of your terminal manual in order to complete this part of SETSCREEn.

NOTE: IF A FUNCTION DOES NOT EXIST ON YOUR TERMINAL, ENTER A <RETURN>!

FUNCTION	EXPLANATION & EXAMPLE
CLEAR SCREEN	Enter the code(s) that clears the screen (separated by spaces if more than one number is to be entered). EXAMPLE: 27 42 REQUIRED FUNCTION
CLEAR TO END OF SCREEN	Enter the code that clears from the cursor to the end of the screen. EXAMPLE: 27 34 1 OPTIONAL FUNCTION
HIGHLIGHTED SCREEN	If your terminal supports highlighting (or another display attribute that you prefer), enter the value that begins highlighted display. Then data in the windows will appear in highlighted mode. EXAMPLE: 27 40 OPTIONAL FUNCTION
NON-HIGHLIGHTED SCREEN	Enter the code(s) to restore your terminal to the normal (low intensity) mode. EXAMPLE: 27 41 OPTIONAL FUNCTION
DESTRUCTIVE BACK SPACE	This code will move the cursor one space to the left, replacing the character with a space. EXAMPLE: 8 32 8 REQUIRED FUNCTION
TERMINAL INITIALIZATION	This code is provided to set your terminal to the proper mode (if required) when DataFlex is loaded. EXAMPLE: 27 40

HOW MANY CHARACTERS WIDE IS YOUR TERMINAL?	(79 is standard)
HOW MANY LINES LONG IS YOUR TERMINAL?	(24 is standard)
USE 7 OR 8 BIT ASCII CHARACTERS?	(7 is standard)

PART FOUR: CURSOR POSITIONING CODES

Part Four of SETSCREEn involves entering the codes that actually position the cursor on the screen. They are entered in the same way as the screen functions above, as decimal numbers separated by spaces. The cursor positioning function will generate a string value that will be sent to the terminal. The following questions will determine how that string is constructed.

```
  POSITION LINE BEFORE COLUMN (Y OR N) Y
  CODES TO PRECEDE LINE COORDINATE:  27 64
```

If your terminal expects the line co-ordinate to precede the column, then answer the question with "Y"; otherwise respond with "N". Next, enter the character string (in decimal, separated by spaces) to preceed the line and column coordinates. If your terminal does not require a character string before one of these coordinates, just enter <RETURN>.

```
  LINE POSITION CODE OFFSET (BIAS) 32       (line 0 is code 32)
  CODES TO PRECEDE COLUMN COORDINATE:

  COLUMN POSITION CODE OFFSET (BIAS) 32     (column 0 is code 32)
  CURSOR POSITIONING TERMINATION CODE:
```

The offset codes will be ADDED to the actual cursor position starting at zero.

Most terminals require the positioning codes as a one byte binary number. For these, answer the question below with a "B". Some terminals (mostly ANSI terminals) require the code as an ASCII number.

```
  SEND CURSOR POSITIONING CODES IN BINARY OR ASCII (B/A): B

  NUMBER OF NULLS TO SEND AFTER CLEAR SCREEN AND CURSOR POSITIONING: 0
```

Some (mostly older) terminals require a delay after a screen operation. For these, enter the number of null fill characters that should be sent.

IS THIS DATA CORRECT (Y/N) Y A "N"o response will allow you to verify your entries or repeat this section in case of errors.

THE SETSCREEN MENU

This ends the initial setup of DataFlex screen codes. If you enter an incorrect code, or want to change a function key, you may re-select any of the options from the menu below. If you re-enter the SETSCREEn program, the initial screen display will list your manually-defined terminal on the pre-defined list under the identifier you established for it.

```
  EDIT:
        (1) SET SYSTEM OPTIONS
        (2) ASSIGN FUNCTION KEYS
        (3) SET TERMINAL CONTROL CODES
        (4) DISPLAY PRESENT CONFIGURATION
        (5) PRINT CONFIGURATION
        (6) END / SAVE CHANGES
        (7) ABORT SETSCREEN - DO NOT WRITE CHANGES
```

From this menu you may select the function which you want to change.

It will generally be useful to select option (5) to print a copy of the configuration as a reference for the FLEXKEYs.

When your configuration is complete, use option (6) to save your changes and return to the DataFlex Menu.

OPERATING TUTORIAL

This tutorial is designed to familiarize you with DataFlex. Its objective is to introduce you to several necessary concepts and functions of the system without going into depth about how the processes are carried out within DataFlex.

It is intended that you use this section of the user's manual in a learning "session", or series of "sessions", actively exploring DataFlex on your system with the manual as your guide.

The Programming Tutorial in the next section of this manual will guide you through the steps involved in creating your own database with DataFlex

SYSTEM CHECKOUT...

Before proceeding with the tutorial, certain prerequisites must first be confirmed to make sure that DataFlex is properly operational on your system. We don't want an improper or incomplete installation to interfere with your introduction to DataFlex.

First, you should have followed the installation instructions packaged with your DataFlex Master diskettes for your particular version of the product. Next, you should have executed the SETSCREEn Program to configure DataFlex for your particular system's CRT control codes and command key setup. Also, you should have printed out a list of your system's command key set up using option 5 on the SETSCREEn Menu to give you a guide to follow during the familiarization. If you have not run SETSCREEn, or have not printed your Flex-Key configuration, do those steps before proceeding further here.

With these items accomplished, you should turn on your system and:

> FOR FLOPPY DISK BASED COMPUTERS, load a diskette containing a copy of the operating system and the DataFlex programs into drive "A:", and the sample data diskette into drive "B:"; OR
>
> ON HARD DISK BASED COMPUTERS, "boot" the system and "log in" to the drive or work area where DataFlex has been loaded from the DataFlex master diskettes.

You should then be able to determine that specific DataFlex files are available on your "logged in device" (see glossary) by having the system display a directory.

If we assume that you are logged in to the "A:" drive (and you may not be -- particularly on a hard disk system), then you should be able to enter the DIR command indicated below (shown in boldface) and have a display containing the listed file names (quite possibly in a different order than displayed here) shown on your CRT screen along with the names of other ".COM" files that may be on the same diskette or hard drive partition (replace ".COM" with ".CMD" on CP/M-86 systems):

```
        A>DIR *.COM  <RETURN>    (or DIR *.CMD on CP/M-86)

        A:FLEX    .COM    COMP    .COM    QUERY    .COM
```

If these file names are displayed, then you have the necessary components in place to proceed with this section. If all of these are not present in the displayed directory, try the command again. If the above files are still not shown, refer back to the installation instructions to make sure that they were completed.

O.K., LET'S GET STARTED...
DataFlex can be run in a completely menu-controlled environment. We will use the sample menu that is delivered with DataFlex as our guide. To get to that menu and begin, type the word FLEX at the command level of your operating system and press <RETURN>.

```
        A>FLEX <RETURN>
```

In a few seconds, the DataFlex Menu should display...

```
 YOUR REGISTERED NAME                                 Serial # : ____
                   .===================================.
                   ||     ---- DataFlex Ver. 2.0 -----    ||
                   ||            MASTER MENU              ||
                   ``===================================''
 .-----------------------------------------------------------.
 |                 1  DataFlex demo files                      |
 |                 2  DataFlex Configuration                   |
 |                 3  System Utilities                         |
 |                 4  Set System Date                          |
 |                 5  Run DataFlex Configuration               |
 |                 6  Enter System Command                     |
 |                 7  Exit To Operation System                 |
 |                                                             |
 `-----------------------------------------------------------'
     |    Please select one of the above options or press    |
     |    <RETURN> to go back one menu >>>>>>>>>>>>>>>> _     |
===============================================================


               ***  DataFlex Menu System  ***
```

The Menu is a DataFlex configuration (named MENU.FLX) that stores the menu option "data" in a data base file. You will learn how to modify the existing menus and put your own configurations on the screen later on. For now we will use the standard menu to begin our tour through DataFlex.

SELECTING A MENU OPTION...
To select an option from the menu, just press the corresponding number AND <RETURN>. Let's start by selecting the DataFlex Demo Menu... press a number one (1) on your keyboard, and another menu will display:

```
 YOUR REGISTERED NAME                                Serial # : ____
              .====================================.
              ||              DataFlex 2.00        ||
              ||            Demo Data Files        ||
              ``===================================''
 .-------------------------------------------------------------------.
 |                   1  Enter Vendors                                |
 |                   2  Enter Inventory Items                        |
 |                   3  Enter Customers                              |
 |                   4  Inventory Movement Report                    |
 |                   5  Customer Report                              |
 |                   6  Vendor Report                                |
 |                   7  ON-LINE Invoice                              |
 |                   8  QUERY Data Files                             |
 |                                                                   |
 `-------------------------------------------------------------------'
      |    Please select one of the above options or press    |
      |    <RETURN> to go back one menu >>>>>>>>>>>>>>>> _     |
===================================================================


                  ***  DataFlex Menu System  ***
```

This menu displays a list of predefined options that you can use to see how DataFlex applications work. Enter a number one (1) to select the VENDOR File Maintenance option. DataFlex will now load and display a screen "image" used for the entry, recall and editing of data in the sample VENDOR data base file.

```
DATA ACCESS CORPORATION                                   VENDOR MASTER FILE
=========================================================================

    VENDOR NUMBER: <___>

          COMPANY: <______________________________>
          ADDRESS:  ______________________________
             CITY:  ______________ ST: __  ZIP: _________

            PHONE: (___) ___-____

     TERMS (DAYS):  ___   CREDIT LIMIT: $______.__
                    OUTSTANDING CREDIT: $______.__
                                        -----------
                      REMAINING CREDIT: $______.__

     TOTAL PURCHASES: $______.__

       LAST PURCHASE: __/__/__

Windows in < > can be used to FIND vendors
```

FLEX-KEY OPERATION...
DataFlex uses "Flex-Keys" to accomplish various actions on a screen display. These are in lieu of a menu which would list the options available to a system operator at any given point in an application. The Flex-Key concept works well because the same keys are used for the same function throughout all DataFlex applications on a given system. For example, the same key is used on all screens to FIND a record in the data base, or to CLEAR displayed information from the screen image. As noted above, Flex-Keys are specified in the DataFlex SETSCREEn Program. Specific detailed information on the use of the Flex-Keys is in this manual in the chapter on Operator Procedures.

"FINDING" A RECORD IN THE DATA BASE...
Your cursor will wait in the VENDOR NUMBER "window" on the screen. Enter a number one (1), and press the FIND Flex-Key on your keyboard (as defined in the Flex-Key list printed with SETSCREEn). The screen should display information for Data Access Corporation. You have just executed your first DataFlex Command!

CLEARING INFORMATION FROM THE DISPLAY SCREEN...
To remove this information from the screen, press the CLEAR ALL Command Key. The screen should be restored to the same status as when it was originally displayed from the menu.

"FINDING" A RECORD BY AN ALTERNATE "KEY"
This screen is configured to allow VENDORs to be found by either the VENDOR NUMBER or the VENDOR NAME. Above, you found a record by VENDOR NUMBER. Now, we can find the same record by VENDOR NAME. While the cursor is positioned in the VENDOR NUMBER window, press <RETURN>, and it should move to the VENDOR NAME window. (The zero which appears in the VENDOR NUMBER window signifies null input, and may be ignored whenever it appears.) With the cursor in the VENDOR NAME window, type the word "DATA" (in capital letters), and press the FIND Flex-Key. Again, the information for Data Access Corporation should be displayed.

MOVING THE CURSOR AROUND THE SCREEN...
Pressing the <RETURN> command key will move the cursor sequentially through the entry windows of the screen image. To move backwards on the screen (for example from the VENDOR NAME window back to VENDOR NUMBER), press the BACK WINDOW Flex-Key. Each strike of <RETURN> or BACK WINDOW will move the cursor forward or backward one window. One caution: a BACK WINDOW from the **first** window, or a <RETURN> from the **last** window will cause all the windows to clear, and possibly a "record" to be saved which you didn't intend to. The safe way to clear all the windows is to strike the CLEAR ALL Flex-Key, and to leave the screen altogether, is to strike the ESCape Flex-Key.

"SCANNING" THE DATA BASE...
DataFlex provides a means for you to scan the records in the data base by any pre-defined key index. You can enter the index at any point and move forward or in reverse through the records in the data base file. With a clear screen image (as described above) and the cursor in the VENDOR NUMBER window, press the NEXT RECORD Flex-Key. Vendor number "1" should be displayed. Now, press NEXT RECORD again... vendor number "2" should be displayed. This same process can be used on any indexed window on a DataFlex screen image. It will not function on a window that has un-indexed data.

A scan of data base information can begin at any place in an index. For example if you wanted to scan all records with a VENDOR NAME that started with the letter "G", you would move the cursor to the VENDOR NAME window and enter a "G". Then press the NEXT RECORD Flex-Key and the first record in the data base which has a VENDOR NAME alpha-betically following the letter "G" will be displayed, possibly "GARNER SYSTEMS" or "GENERAL ELECTRIC". To scan to the record following the one displayed, press NEXT RECORD again, and again, etc. To move in reverse, use the PREVIOUS RECORD Flex-Key.

ENTERING A NEW RECORD INTO THE DATA BASE...

So far everything that we have done has used data already stored in the data base. It's time for some of your own information! Clear the screen using the CLEAR ALL Flex-Key. Next, press the <RETURN> Key and a zero (0) will be displayed on the screen temporarily; when your new data is saved a VENDOR NUMBER will automatically be assigned to your new entry.

The cursor should now be in the VENDOR NAME window. Type the name "MICRO MANUFACTURING" and press <RETURN>. Then fill in a fictitious address, city, state, ZIP, area code and phone number, terms and credit limit. When the credit limit has been entered, your data for MICRO MANUFACTURING will be saved in the data base automatically. The remaining windows on the screen contain calculated data that is up-dated by other parts of the system. No operator entry can be made to those windows.

The new record is now stored in the data base. To FIND it, place the cursor in the VENDOR NAME window, type MICRO MANUFACTURING and press the FIND Flex-Key. MICRO MANUFACTURING's record should be displayed on the screen. You can CLEAR the screen image, or use NEXT or PREVIOUS RECORD Flex-Keys to scan other records in the data base.

CHANGING INFORMATION IN THE DATA BASE...

One of the most important functions of any information system is to provide for the on-going updates to stored data. This would include such examples as address changes in a mailing list, price changes in an inventory system, etc. The process of making these changes in DataFlex is easy... you FIND the record to be changed or updated, enter the necessary changes on the displayed screen image, and then SAVE the record back to the data base. Let's try it...

FIND the data base record for MICRO MANUFACTURING. Move the cursor down to the phone number windows and change the existing data to something else (e.g. (414)354-6868 to (939)456-9876). Now press the SAVE Flex-Key, and the record will be stored back in the data base with the changed phone number. FIND MICRO MANUFACTURING again to verify that your change has been recorded.

The SAVE Command can be executed at any time that a record is displayed on the screen image. The displayed information will be SAVEd regardless of the position of the cursor in the image. This allows you to save keystrokes by moving the cursor as far into an image as necessary to change information, and then execute the SAVE without moving to the end of the screen.

HOW TO EXIT THE SCREEN & RETURN TO THE MENU...

To leave any DataFlex screen image and return control to the Menu, simply press the ESCAPE Flex-Key. The screen image will clear and in a moment the DataFlex Menu will re-display ready for another selection.

CHECKPOINT...

SO FAR, we have explored:

- -- Loading DataFlex from the operating system
- -- Making a selection from the DataFlex Menu
- -- Displaying a screen "image"
- -- FINDing information in the data base
- -- CLEARing a screen "image"
- -- Moving the cursor around the screen image
- -- Scanning data base information
- -- SAVEing new information in the data base
- -- Changing existing information in the data base
- -- Exiting from a display screen to the menu.

If you are comfortable with the operation of the system at this point, proceed to Phase II to build your own sample application file. If there are areas of this section which are unclear, or if you just want to develop more of a familiarity with the various operations, you can review any of the preceding operations until you feel comfortable enough to move on to the next phase.

THIS PAGE INTENTIONALLY LEFT BLANK

QUERYING THE DATABASE

OVERVIEW

The DataFlex QUERY Program enables any level of system user to quickly and easily extract information from a DataFlex database. QUERY can automatically format the file data for reports, screen displays, or disk files. The information can be output by any index, and can be selectively extracted according to specifications entered at run time. The operation of QUERY is completely interactive and non-technical.

Additional features of the QUERY Program include optional totalling of numeric fields, and a full range of logical selections (less than, less than or equal to, etc.) can be used. Up to ten selections per session are allowed.

Output from QUERY is device-independent, meaning that it can be directed to the screen, printer or a disk file. If QUERY's output is directed to a disk file, the data is stored in ASCII format so that it can be edited or read by other programs. QUERY also offers the option, once a successful QUERY has been completed, to save it to disk as a report configuration source file which can be compiled and run as a DataFlex program, repeating the QUERY as many times thereafter as may be desired. Alternatively, the QUERY-produced file may be further modified to become any other kind of report that may be desired.

Although QUERY can handle only one DataFlex database file at a time, prompted multifile data base inquiries can be created as DataFlex configurations by use of the REPORT command macro, described in another section of this manual.

PROGRAM START-UP

QUERY is generally started from one of two levels of the system: a menu selection or the operating system command mode.

If the DataFlex MENU system is installed and active on your system, enter the number corresponding to QUERY on the Data Flex Demo File

Menu or other menus on which it may be included on your system. For example, choice number 8 in the Demo menu below:

```
                           DataFlex Demo Files

                         1 Enter Vendors
                         2 Enter Inventory Items
                         3 Enter Customers
                         4 Inventory Movement Report
                         5 Customer Report
                         6 Vendor Report
                         7 ON-LINE Invoice
                    -->  8 Query data files
```

If the menu system is not in use, QUERY may be invoked from the operating system by simply typing the program name:

```
        A>QUERY
```

If the file number assigned to the DataFlex data file is known, QUERY can be given that number on the operating system command line and the step of displaying a list of files available for query will be skipped. E.g.:

```
        A>QUERY 21
```

If no file number accompanies the entry to QUERY, the program will display a list of files available on the system, and prompt you for a selection. Enter the number of the file to be queried.

```
                 File       File
                 Number     Name
                 ------     ----------
                    21:     VENDOR
                    22:     INVENTORY FILE
                    25:     CUSTOMER FILE
                    29:     LINE ITEMS




          ENTER FILE NUMBER, OR RETURN FOR MORE FILES: 21
```

PROGRAM OPERATION

When a file has been selected, QUERY will display the names of all of the fields in the selected data file. For example:

```
     0: RECORD #          1: VENDOR           2: ADDRESS
     3: CITY              4: STATE            5: ZIP
     6: PHONE_AC          7: PHONE_EXCHG      8: PHONE_NUMBER
     9: TERMS            10: CREDIT_LIMIT    11: CREDIT_USED
    12: PURCHASES        13: LAST_PURCHASE


          SELECT THE ITEMS THAT YOU WANT TO BE QUERIED FROM
          DATA FILE #21<VENDOR>
          WHEN YOUR FIELD SELECTIONS ARE COMPLETED, PRESS <RETURN>
          TO CONTINUE ON TO THE NEXT SECTION OF QUERY.

    TOTAL COLUMNS = 10          ENTER FIELD # TO PRINT: 2
```

ENTER FIELD # TO PRINT: is where you respond with the number of the field from the displayed list in the order you wish the information to be displayed or output. Your entries should be: field #, <RETURN>, field #, <RETURN>, etc. If you output your query in a report format, the columns and the column headers in your display or printout will be the field names which you select here.

The COLUMNS ALLOCATED prompt will accumulate the number of horizontal columns (characters) used as you select fields for formatted output. The number of columns are calculated by taking the larger of: the size of the field you entered from the list, or the size of the data field name as it appeared on the list plus a space separator.

You may enter as many fields to be included in the QUERY as you want. However, keep the following in mind: the standard video display screen and some printers can only accommodate 80 columns (some actually accommodate only 79). If the COLUMNS ALLOCATED exceeds 80 (or 79), the screen will "wrap around" which may make the line of unformatted data unreadable.

Many printers will accommodate 132 columns or more (e.g. printers capable of printing at 12 characters per inch or more). QUERY can create up to 255 characters of output per line.

As you respond to the prompt line, QUERY will mark your field selections by placing an asterisk in front of the field tag name. Should you change your mind about a field that has been selected for output, cancel the last selection by using the BACK WINDOW command key.

When all desired fields have been chosen, press the <RETURN> key to move on to the next section. QUERY will ask if totals are wanted on the numeric fields.

```
        DO YOU WANT TOTALS PRINTED AT THE END OF THE QUERY? <N>
```

The next prompt will be:

```
        ENTER ITEM TO SELECT BY OR PRESS <RETURN> TO CONTINUE:
```

MAKING SELECTIONS IN QUERY

These are the logical operators available in QUERY (less than, less than or equal to, etc.), that are used to define a subset of data from a file.

```
        <: LESS THAN                  [: LESS THAN OR EQUAL TO
        >: GREATER THAN               ]: GREATER THAN OR EQUAL TO
```

if the field selected is numeric you will see:

```
        NUMBER:
        =: EQUAL TO                   X: NOT EQUAL TO
```

if the field selected is alphanumeric you will see:

```
        STRING:
        $: STRING COMPARE             @: STRING INCLUDES
```

As you respond to the prompt line it will change by translating your responses into the appropriate words. QUERY will also mark the field names list on the upper portion of the screen with the selected symbol. If a greater than and a less than are specified for the same field, a range of records between the two values will be selected. The maximum number of selection criteria allowed is 10.

CHOOSING AN OUTPUT SEQUENCE FOR QUERY

QUERY will display the available indexes for the file. Select the index that you want to determine the sequence or order of the QUERY output. If you press <RETURN> and do not enter an index number, the output will be ordered according to the record number of the data.

```
     1 ---> <VENDOR ID>
     2 ---> <VENDOR NAME>

          OUTPUT OF THIS QUERY SHOULD BE LISTED BY:
```

QUERY will then ask for the TITLE OF THE REPORT. This information will appear at the top of each page of a report format QUERY.

```
          ENTER THE REPORT HEADER:
```

Next, QUERY will prompt for the output destination. At this point you also have the option of using QUERY to create a configuration file for a DataFlex report. The default for this prompt is to output a report formatted query to the screen.

```
          AT THIS POINT YOU MAY SELECT WHERE THE OUTPUT OF THIS QUERY
          SHOULD BE DIRECTED.  YOU MAY ALSO SELECT THE FORMAT FOR
          A REPORT IMAGE FILE.

          OUTPUT QUERY TO:
          (S)creen, (P)rinter, (F)ile, (R)eport image, or (A)bort:
```

An entry of "S" will cause the output to go to the CRT screen; "P" will send the output to the printer; "F" will direct it to a disk file (you will be prompted for a file name); or "R" will create a "Report Image" (see next paragraph). A <RETURN>, or default, response will be executed as though "S" were selected. Both the "F" (file) and "R" (report configuration) options will trigger the questions below.

USING QUERY TO GENERATE A REPORT CONFIGURATION

By making an "R" response to the output option prompt you will start the creation of a report configuration file. The next prompt will ask which of three formats, Report, Read, or Quoted, you want used for the data output when the report is run:

```
          SELECT THE OUTPUT FORMAT TO BE USED
          1)Report, 2)Read, or 3)Quoted format:
```

1) REPORT FORMAT: A formatted columnar report consisting of a title section, column headers, data elements arranged in columns under their appropriate names, and totals (if selected) at the end.

2) READ FORMAT: A sequential ASCII format consisting of each data element on a separate line terminated with a carriage return. This is the format that the DataFlex utility READ uses to "Read" data into a DataFlex data base file.

3) QUOTED FORMAT: A sequential ASCII file format compatible with many versions of BASIC and the "merge" files created and used by many word processors, in which quotes and commas ("b̸,") are used to delimit fields and each record is terminated by a carriage return. Leading and trailing blanks are trimmed from the fields of output from QUERY, but not from output of the generated report configuration (unless you subsequently edit it to do so).

Then you will be prompted:

```
        ENTER THE REPORT IMAGE NAME (8 CHARACTERS MAX)?
```

Respond with the filename up to eight (8) characters in length that you wish the report source configuration file to be stored under by the operating system. QUERY will add an extension of .RPT to your entry resulting in a final name of the file of filename.RPT. Therefore, you must not specify your own extension to the filename. If you do, QUERY will ignore it.

If a file already exists with that name (filename.RPT), QUERY will issue a warning message to that effect and ask whether or not to overwrite the file. The default response in this case is <N>.

Next you will see a message:

```
        REPORT IMAGE CREATED!
        PRESS <RETURN> TO CONTINUE......
```

QUERY will then again display the output option prompt shown above, which is intended to give you the opportunity to output the actual data (as opposed to the report configuration file which you have just created). If your only purpose for running QUERY at this point was to create the REPORT configuration file, choice "A" to abort would be your choice.

In order to use the report configuration file created by QUERY, you must compile it into a .FLX file with the command:

```
       A>COMP filename.RPT
```

where filename is the name you gave the file in answer to QUERY's prompt above. Prior to compiling it, you may wish to modify it in one way or another, in which case you may do so as you would with any other DataFlex configuration source file (see the chapter on How to Write a Configuration in this manual). The compiled configuration would then be executed from the operating system with the command:

```
       A>FLEX filename {output destination}
```

PROGRAMMING TUTORIAL

CREATING A DATAFLEX APPLICATION OF YOUR OWN

In this section of the tutorial, you will experience how easy it is to create an application using DataFlex without ever having to "program". Then, once you have an application functioning, you will be introduced to what can be done by putting just the slightest effort into "customizing" your application requirements.

To achieve this, we have decided to appoint you as manager in a company that has just bought their first microcomputer with word processing and DataFlex. As the member of the group who "took a computer course in college", you have been selected to be the one to become familiar with the new system and to undertake the development (and perfection!) of some new management information systems.

Being a creative and thoughtful group, you and your associates recognize the value of properly planning the information system. The first task defined is to develop a personnel information system for the company.

DataFlex provides several methods to implement your application requirements. The easiest and most efficient at this point is the facility for you to create an "image" of a data base file on your CRT screen using a word processor or the DataFlex Editor program.

An "image", in DataFlex terminology, is a format into which data can be inserted for output. It can be displayed on a CRT screen, or on a printed report. The image can include prompting or labelling for information which is stored in the data base, and "windows" where the stored data will be output. DataFlex images are created with a text editor (one is included with DataFlex) or any word processor that can create and edit a plain ASCII file. Creating an image with your editor allows you to develop a visual approach to designing your applications that combines how it looks with how it works.

When an "image" is complete, it may be processed by a utility program supplied with DataFlex, called AUTODEF, into a data base file definition and data entry screen. AUTODEF reads in the "image" file, asks you some questions about how you want things to work, and then outputs another file (called a "configuration source" file) onto your disk that can be compiled and executed as is, or modified to meet a particular set of requirements, and then compiled. The rest of this section illustrates this process. There are other, more flexible ways of accomplishing these functions (such as through FILEDEF) which are described elsewhere in this manual.

Getting Started...

To speed the familiarization process, we have made up an image file and printed it below to help you get underway with the task defined above. We will be using it as the basis of the rest of the tutorial. You can also use it as is or modify it with your word processor or the DataFlex Editor to suit your own tastes for actual use in your own business.

Type it in with your word processor or the DataFlex text editor and save it under the name PEOPLE, with no extension. It should look like this:

```
/SCREEN
.......................... PERSONNEL FILE ..............................

EMPLOYEE LAST NAME : _______________   FIRST: __________   MI: __

          ADDRESS1 : _________________________
          ADDRESS2 : _________________________
              CITY : _______________   ST : __     ZIP : _________

       SOC. SEC. # : _________

        DATE HIRED : __/__/__

          PAY RATE : $_____.__

          PAY TYPE : _  (H=Hourly or S=Salaried)
/*
```

Refer to the PEOPLE "image" file as we learn some basic DataFlex terminology and concepts.

The "/SCREEN" on the first line above is a PAGENAME by which we can address the image with DataFlex commands. The "windows" (see below) are assigned numbers in a left to right, top to bottom sequence. A given window is addressed by the PAGENAME followed by the number of a window, for example the EMPLOYEE LAST NAME window above is SCREEN.1.

The underline characters (___) that you see above (and in the PEOPLE file on your disk) are called data "windows". They represent the area on a CRT screen where information can be entered or displayed. The number of underline characters in a window determines the length of the data that is valid for entry or display. Ten underlines define a window that will accept ten characters of data input or display.

PEOPLE illustrates the three types of data that can be stored by DataFlex. The three types are called ASCII, NUMERIC and DATE.

ASCII data is any character or group of characters that can be typed from the keyboard: the letters A to Z and a to z; numbers 0...9; punctuation such as comma (,), period (.), colon (:), etc.; and space. That's right... a space character, generated when you press the space bar on the terminal keyboard, is just as important to the computer as any other character.

Examples of ASCII data are: the name of a city (MIAMI), a person's name (Mike), or a single character symbol ("N" for new or "U" for used). Numbers can be entered into ASCII windows, but they are treated only as characters and have no numeric value or characteristics. An example of numbers that could be carried as ASCII data would be identification numbers, such as the registration number on the side of an airplane (N8847J).

ASCII character data is represented in the image by underline characters (_____) such as that following the employee name in PEOPLE.

By placing certain other characters among the data "window" underline characters, you can specify either of two other types of data for the window.

NUMERIC data windows are specified by the inclusion of one decimal point (.) in place of one of the underline characters for a window. They are appropriate for data which signifies quantities, such as amounts of money, volumes of fuel, or quantities of parts in inventory, particularly if calculations are to be done on the data. NUMERIC data windows will accept entry only of numbers, the minus sign (-), and the decimal point (.).

If the decimal point is at the rightmost position in a numeric data window (____.), the window will accept entry of whole numbers only (9999 but not 9.99). If the decimal point is placed within the group of underline characters that make up a numeric window (_____.__), then it formats the entered or displayed data according to the position of the decimal point (the window _____.__ would display 12.3 as 12.30).

DATE windows are shown in the image by underline characters and slashes (/) in the specific format "__/ __/__". The "Date Hired" in the PEOPLE "image" is an example of a date window.

The placement of data windows, and any descriptive prompting or other information is not of importance to DataFlex or the AUTODEF Utility. Only the three different types of data windows are of interest to DataFlex. Don't overlook the "/*" at the bottom of the image. It's very important, as you will see.

Using your word processor or the DataFlex Editor, you can inspect the PEOPLE image file at your leisure... change it, lengthen one of the ASCII windows (just add some underlines), translate the prompting to a foreign language, etc. By doing this you can get a feel for how an "image" can be manipulated to be just the way you want it.

Enough about how to do all of this, let's get it done! Load DataFlex from the operating system by typing FLEX. The Master Menu will display. Select the DataFlex Configuration option from the Master Menu. On that menu, select the option for AUTODEF.

The CRT screen will clear and prompt:

```
    ENTER <RETURN> TO EXIT OR
    "ROOT NAME" OF FILE DEFINITION TO CREATE:
```

If you want to return to the menu, press <RETURN>, otherwise type PEOPLE to let AUTODEF know the name of the file that you want to work on. If you are working on a system where the file PEOPLE is not located on the "logged in drive", then you must type the drive designation where the file is stored, for example, "B:PEOPLE" would be a typical entry for a floppy disk based system where the PEOPLE file is stored on drive B:.

DataFlex will now display a list of the files created as a result of AUTODEF's processing of the PEOPLE file:

```
    FILENAME SUMMARY:
              IMAGE NAME          d:PEOPLE
              FILE ROOTNAME       d:PEOPLE
              DATAFLEX FILE NAME  d:PEOPLE
              CONFIGURATION FILE  d:PEOPLE.FRM
```

The "d:" in the above list will correspond to any drive designation that you may include with PEOPLE (e.g. C:PEOPLE).

DataFlex will then prompt:

```
    WHAT IS THE MAXIMUM NUMBER OF RECORDS "PEOPLE" COULD HAVE?
```

For the purposes of our work here, a small number of records will be sufficient... enter a value of 100.

Without further interaction, AUTODEF will read the PEOPLE image file from the disk. When the image has been read, you will be prompted for the names of the data windows on the image. Any name up to 15 characters which begins with a letter and contains no spaces can be used, but for purposes of consistency as we go through this exercise, please enter the following list of window names as they are requested.

LNAME	for Last Name
FNAME	for First Name
MI	for Middle Initial
ADDR1	for the first line of Address
ADDR2	for the second line of Address
CITY	for City
ST	for State
ZIP	for ZIP Code
SSAN	for Social Security Account Number
DATE	for Date Hired
PAYRATE	for the Pay Rate
PAYTYPE	for the Pay Type

Each window will be highlighted by asterisks (*****) in place of the underline character (_____) as you are being asked about it.

The example below indicates the appearance of the screen when it is prompting for the Last Name:

```
........................ PERSONNEL FILE .............................

EMPLOYEE LAST NAME : ***************   FIRST: __________    MI: __

          ADDRESS1 : _________________________
          ADDRESS2 : _________________________
              CITY : _______________     ST : __  ZIP : _________

       SOC. SEC. # :   _________

        DATE HIRED :   __/__/__

          PAY RATE :  $_____.__

          PAY TYPE :   _ (H=Hourly or S=Salaried)


ENTER FIELD NAME FOR FIELD_1:
```

As you proceed, if you wish to back up and rename a window, the BACK WINDOW Flex-Key will move the asterisk "cursor" back to the desired window. After all windows have been named, you will have the opportunity to redo the process if any mistakes have been inadvertently entered. The prompt is:

```
    Press "C" to continue, or "R" to re-do this page: _
```

If the above list has been properly entered, press "C"; if not, press "R" to re-enter the window names.

When "C" is pressed, a list of the window names will be displayed for you to make a selection of the fields which are to be used to FIND the data in the data base.

```
    1. LNAME      2. FNAME    3. MI        4. ADDR1
    5. ADDR2      6. CITY     7. ST        8. ZIP
    9. SSAN      10. DATE    11. PAYRATE 12. PAYTYPE


    ENTER FIELD NUMBER TO INDEX OR <RETURN> TO END: __
```

Since we will want to find personnel by Social Security Number, enter "9" to select that window for indexing.

DataFlex can allow for duplicate entries in a data file, or it can reject them. The choice is up to you. In the case of a name window, you would probably want to allow for duplication, whereas in the case of a social security number (which is unique for each person) a duplicate entry (two identical social security numbers constitute an erroneous entry) should be rejected.

The next prompt establishes whether or not duplication is allowed:

```
    WILL THE DATA IN THIS INDEX BE UNIQUE? <N> _
```

If you want duplication allowed, press "N", if you want to reject duplicate entries, press "Y".

Multiple data windows can be used to create a single index to FIND information in the data base. The three fields containing data on an employee's name are a good example of applying this feature. By combining the LNAME, FNAME and MI windows, a single index will put employees in proper alphabetical sequence. That is, they will be in order by Last Name, then First Name, then Middle Initial.

To enter multiple data windows in one index, place the numbers of the windows in a single index entry, separated by spaces. For the example at hand, that entry would be:

```
    ENTER FIELD NUMBER TO INDEX OR <RETURN> TO END:  1 2 3
```

Make the above entry to create the second index for the file being created. Then, designate that the data in this index will NOT be unique (allow duplicate names).

After you have entered the index information for PEOPLE, AUTODEF will process the file definition and create the index and data files for the operation of the application.

A command file was created by AUTODEF called PEOPLE.FRM. It is located on the same disk drive where the PEOPLE image file is located. It contains the image that we have been working with, and DataFlex Commands that make the image function with the data file that was created by AUTODEF. PEOPLE.FRM looks like this:

```
/SCREEN
.......................... PERSONNEL FILE ...........................

EMPLOYEE LAST NAME : _______________   FIRST : __________   MI : __

          ADDRESS1 : _________________________
          ADDRESS2 : _________________________
              CITY : _______________   ST : __   ZIP : _________

       SOC. SEC. # : _________

        DATE HIRED : __/__/__

          PAY RATE :$_____.__

          PAY TYPE : _ (H=Hourly or S=Salaried)

/*
OPEN PEOPLE
ENTER PEOPLE
  ENTRY PEOPLE.LNAME
  ENTRY PEOPLE.FNAME
  ENTRY PEOPLE.MI
  ENTRY PEOPLE.ADDR1
  ENTRY PEOPLE.ADDR2
  ENTRY PEOPLE.CITY
  ENTRY PEOPLE.ST
  ENTRY PEOPLE.ZIP
  ENTRY PEOPLE.SSAN
  ENTRY PEOPLE.DATE
  ENTRY PEOPLE.PAYRATE
  ENTRY PEOPLE.PAYTYPE
ENTEREND
ABORT
```

PEOPLE.FRM can be loaded with the DataFlex Editor or your word processor and inspected.

PEOPLE.FRM must now be "compiled" to be run under DataFlex. The Data-Flex compiler reads PEOPLE.FRM in the form that we see it here, and processes it into a highly compressed file that contains the internal codes and instructions on which DataFlex operates. Should we wish to change the way that the personnel application works (and we will), PEOPLE.FRM can be modified with your editor and re-compiled. After processing PEOPLE.FRM, the compiler will output a file called PEOPLE.FLX.

COMPILING A DataFlex CONFIGURATION FILE

To compile PEOPLE.FRM into an executable DataFlex program, select the DataFlex Configuration option from the Master Menu screen. On the Configuration Menu, enter the number for the Compile a Configuration option. When prompted there for the file name to compile, type PEOPLE.FRM. Make sure to include the .FRM file extension or you will be instructing the program to compile the file PEOPLE, which does not contain the necessary command lines to create an application.

Your disk drives will now become active, and some recognizable portions of PEOPLE.FRM will be displayed on the CRT screen. Don't be alarmed! You may think that what you see is some form of "programmer's revenge". In fact, it is a display of the command lines as DataFlex is compiling them and serves some very useful functions to advanced DataFlex users.

The file creation process, done with AUTODEF in the case of this example, creates a support file required for the compiler. The file has a name of filename.FD where "filename" is derived from the name of the image file processed by AUTODEF. The image file name was PEOPLE. Therefore, a file called PEOPLE.FD now exists on the disk.

When compilation is complete, you will be returned to the DataFlex Menu. You are now ready to test your first DataFlex application!

From the Configuration Menu or Master Menu, select the "Run a DataFlex Configuration" Option. When the option is selected, you will be prompted for the name of the application that you want to run. On a floppy disk system, enter the drive letter of the diskette drive on which PEOPLE is located followed by the word PEOPLE, e.g. B:PEOPLE. On a hard disk system, no drive letter specification is necessary unless the PEOPLE file group is located on other than the logged-in device. See the menu sample on the next page. It shows the RUN option selected from the menu, and an entry to RUN the file B:PEOPLE(.FLX) has been made.

```
 YOUR REGISTERED NAME                                      Serial # : ____
                  .===================================.
                  ||             DataFlex 2.00          ||
                  ||        Configuration Utilities     ||
                  ``===================================''
 .----------------------------------------------------------------.
 |                1  File Definition                               |
 |                2  Menu Definition                               |
 |                3  System Editor                                 |
 |                4  AUTODEF (quick entry screen)                  |
 |                5  Compile A Configuration                       |
 |                6  Run a DataFlex Configuration                  |
 |                7  File recovery (frel)                          |
 |                8  Re-index a file                               |
 |                9  Generate ASCII file READ                      |
 `----------------------------------------------------------------'
     |    Please select one of the above options or press     |
     |    <RETURN> to go back one menu >>>>>>>>>>>>>>>> 6     |
=================================================================
You have selected to run a DataFlex configuration
Please enter the name of the configuration..........   B:PEOPLE
               ***  DataFlex Menu System  ***
```

In a few seconds, your screen should look like the PEOPLE image that we have been dealing with. But now it is more than just an image; it is an active data entry screen that is waiting for you to enter data into, and recall data from the data base.

ENTERING INFORMATION ON A DataFlex SCREEN IMAGE

Your cursor should be positioned in the leftmost position of the last name window. To enter information into the system, you simply "fill in the blanks" in the image. Enter a last name, first name, middle initial, address, etc. until all of the windows contain data. If there is not any data to be entered into a window, press the RETURN key and the cursor will jump to the next window.

When data (and a <RETURN>) has been entered to the last window (Pay Type in the case of the PEOPLE screen), the displayed information will automatically be stored in the data base on the disk. It's that simple. Now, enter another two or three people in the same fashion... it will be useful to have a few records stored in the data base as we proceed.

RECALLING INFORMATION ON A DataFlex SCREEN IMAGE

Having saved a record in the data base, the screen image should be clear and your cursor should be located in the leftmost position in the last name window. Press the FIND Command Key. The personnel record for one of the employees that you just entered on the screen should appear. Which one is it?? If all is functioning properly at this point, the displayed name should be the one alphabetically at the beginning of those that you entered.

Leaving the cursor in the same place in the last name window, press the NEXT RECORD Command Key. The record next in alphabetical sequence should now fill the screen windows. This process can be repeated to scan through the list of all personnel until the end of the file is reached. Now try the PREVIOUS RECORD Command Key. Catching on??

You can directly FIND any name in the data base just by entering it in the name window and pressing the FIND Command Key just as with the VENDOR screen image covered in PHASE I of the tutorial. To locate George Jones' record (assuming that one had been entered for him), type "Jones" into the last name window and "George" into the first name window. Then press the FIND Command Key. George Jones' record will be found in the data base and displayed on the screen instantly.

If you were looking for a person named "Jones" but did not know his first name, you could enter just the last name and press FIND. Data-Flex would find the first Jones in the data base and display his record. The "first Jones", if there were multiple "Joneses" would be a person named Jones with the alphabetically lowest first name. After finding the "first Jones", you could use the NEXT RECORD Command to move further through the data base to find and display the record for the Jones that you were seeking.

If you wanted to display the record for a person and could not remember the exact spelling of his name, enter something close, or the first part of the name. For example, if you wanted to display the record for McDougal, you could enter "McD". DataFlex will then find the record closest to the data that you have entered and display it. If the displayed record is not the person that you were looking for, (there might be a McDonald in the data base who would be found if you entered "McD") you can use the NEXT RECORD or PREVIOUS RECORD Command Keys to scan up and/or down the data base until the correct personnel record is found and displayed.

Impressed? You should be. You've done in a matter of minutes what it could have taken an experienced programmer at least days to complete with conventional software tools. Good job!! And we've only begun to peek at the power of DataFlex!

Remember that in AUTODEF you were asked which of the data base fields were to be used to find records in the data base? You selected the name fields and the social security field. To find a record by social security number, press CLEAR ALL to remove the currently displayed record, then press RETURN to move the cursor down to the social security number window. If you should move the cursor too far, press the BACK WINDOW Command Key to move the cursor backwards in the image.

With the cursor in the social security number window, you can enter a known number and press FIND to display the record. Or, just press the FIND Command Key to recall the record of the individual with the lowest social security number. The NEXT and PREVIOUS RECORD Command Keys will now cause a sequence of records to be displayed, in order, by social security mumber.

CHECKPOINT...

So far, you have never left the displayed DataFlex data entry screen. You stored records in the data base alphabetically by simply typing data into the windows on the screen, and then recalled and displayed the information directly or in two different sequences (name and social security number) with single key-stroke commands.

CUSTOMIZING YOUR DataFlex APPLICATION...

As good as this system for personnel information is (given the time that you've spent on it), let's get picky... how could it be better? Where could it be improved?

Let's start with appearance of the screen itself. Is the general layout of the information easy to read? ...is the information in logical order? We shall accept that it is.

However, you have designed and built it, and have a knowledge of its inner workings. The new person in the office who will be doing most of the data entry of the personnel information will not have the knowledge that you do. What might that person need or want to know?

Some suggestions: How about a help message about finding stored records? ...or a message explaining how to exit the screen and return to the menu? Displaying information on the screen with the data will make the operation of the system more "friendly", and reduce the number of questions that might have to be answered as different people use the computer based personnel information files.

Changes of this type refer only to the application, not to the structure of the information that is stored in the data base. Therefore, we need only to modify the PEOPLE.FRM file since it controls the function of the personnel application. Once our modifications are made, PEOPLE.FRM can be recompiled and the application will have new features as a result of the changes.

The modifications are made to PEOPLE.FRM with the text editor. To help identify which windows can be used to FIND information in the data base, you can place some kind of symbol around the name and social security number windows in the screen image. Messages to the operator about how the image works can be simply typed at the bottom of the image for reference. These alterations of the image change the way that the application LOOKS.

Other modifications can be made in PEOPLE.FRM to the commands themselves. New commands can be added, or existing commands can have operational characteristics added to them to change the way that the application **WORKS**.

To proceed with these modifications, load the PEOPLE.FRM file with your text editor or word processor and make the additions to it that are shown in **boldface** in the file on the next page.

```
/SCREEN
........................... PERSONNEL FILE ........................


                             Last              First       MI
     EMPLOYEE NAME  :  <______________ __________ __>

          ADDRESS1  :  _________________________
          ADDRESS2  :  _________________________
              CITY  :  _______________  ST : __   ZIP :  _________

       SOC. SEC. #  :  <_________>  (NO DASHES!)

        DATE HIRED  :  __/__/__

          PAY RATE  :  $_____.__

          PAY TYPE  :  _ (H=Hourly or S=Salaried)

Enter a Name or Soc. Sec. # and press FIND to display record,
OR press <ESC> to exit and return to the DataFlex Menu...
/*
OPEN PEOPLE
ENTER PEOPLE
  ENTRY PEOPLE.LNAME                                  // double slashes allow
  ENTRY PEOPLE.FNAME                                  // you to put comments
  ENTRY PEOPLE.MI                                     // into the command area!
  ENTRY PEOPLE.ADDR1
  ENTRY PEOPLE.ADDR2
  ENTRY PEOPLE.CITY
  ENTRY PEOPLE.ST
  ENTRY PEOPLE.ZIP
  ENTRY PEOPLE.SSAN
  ENTRY PEOPLE.DATE
  ENTRY PEOPLE.PAYRATE
  ENTRY PEOPLE.PAYTYPE {CAPSLOCK,CHECK="HS"}//forces entry to
                                              //uppercase and accepts
                                              //only letter H or S

ENTEREND
ABORT
```

When you have made these changes to the PEOPLE.FRM file, save your edited file to disk, recompile it, and execute again from the menu to see the results of the changes.

Several other enhancements could be included here for PEOPLE. For example, a range check could be applied to the Pay Rate entry to make sure that it did not exceed a certain value, or State could be set to accept only the state in which your business resides if you have no out-of-state employees. The possibilities are almost endless, and that's what DataFlex is all about--fulfilling the almost endless range of application requirements that are encountered in information processing. Good luck!

THIS PAGE INTENTIONALLY LEFT BLANK

FORMATTING WITH IMAGES

OVERVIEW

Throughout DataFlex, formatting can be defined with "IMAGES". This means that wherever possible, your data record, entry screen or report format can look exactly like what you create on the CRT screen with your word processor or text editor. As straightforward a concept as this is, very few products employ it. Dataflex can use IMAGE FORMATTING to create a file definition, create a configuration for data entry and provide the basic format for reports.

The image format should look like a form. It should have a heading, descriptive material for the data to be entered, procedural explanations and prompting, and a place for the data to be entered. The placing and content of the descriptive part of the screen image is totally up to you. It should appear familiar to the operator, and have a logical flow of the data entry fields. The places where the data fields are to be entered into the form image are called DATA WINDOWS. The placement, format and length of the data windows is very important to how DATAFLEX interprets the image.

IMAGE "PAGES"

A single screen image is called a "PAGE". A configuration can have multiple PAGEs, including tiered entry screens, help screens, and any other display that may be desired. Each PAGE can be manipulated individually by the configuration. Image PAGEs can be displayed on the screen, used for data entry or output to a device (printer, console or file). Each PAGE is identified by a page name. The page name goes after a "/" in the first space on the first line of the image, and must begin with a letter and may not contain spaces. A configuration which does not begin with a "/" as the first character in the first space on the first line, followed by a pagename, will be rejected by the compiler. A PAGE continues in an image until the start of the next PAGE (another "/pagename") or the end of the image section (indicated by "/*" at the start of the last image line). Image PAGEs used for screen input/output should not have more lines than the height of your terminal as defined in SETSCREEn (often 24 lines). Pages used for print reports should not have more lines than the paper (labels, cards, etc) that the report is printed on.

The image below defines the last PAGE of a configuration (note the "/*" at the bottom). The pagename is "PERSON" (note the first line).

```
/PERSON
                  MY FIRST SCREEN IMAGE

   FIRST NAME: __________   LAST NAME: __________________

   AGE: __.  WEIGHT: ___._  DATE OF BIRTH: __/__/____
/*
```

The three DataFlex data types (ASCII, NUMERIC, DATE) each have a specific format that can be used for IMAGE data windows.

1. ASCII windows are composed of underline characters (_____). The length of the window (and the data which can be entered there) is defined by the number of underlines. Any printable character can be entered in an ASCII window, one for each underline in the image.

2. NUMERIC windows are defined by placing a decimal point in or at the end of a window (__.__ or __.). If the decimal point is placed at the end, that place will count as a numeric digit position for input (i.e. DataFlex will convert it to an underline when the screen is ready for input), otherwise, the decimal point will display as a decimal position marker only, and NOT count toward the total length of the number. Only numbers, "." and "-" can be entered into a numeric window. Entries containing other characters will be rejected.

3. DATE windows are defined by the format: __/__/__. The date must be entered in the standard MM/DD/YY format as covered previously under "DATA TYPES", unless your system was supplied in "Euro-date" format. Internally, dates are stored as 3-byte numbers representing the number of days from January 1, Year 0 to the date entered (referred to as "Julian dates". For applications involving dates outside the range 1901-1999, dates can be formatted using a 4-digit year (__/__/____). If this format is specified, however, the entire application should use this format consistently, since 12/24/44 generates a different value from that generated by 12/24/1944. 2-digit year entries (which the window will still accept as being in the first century A.D.) should be trapped in the configuration. In 4-digit year mode, DataFlex will handle dates from 1 A.D. through the year 2500. For convenience, the operator may use periods "." or any other punctuation mark in place of the diagonals "/" in entering dates. Internal calculations on screen and report displays are not affected by use of this option.

Data windows are referenced by number within a PAGE, with the window numbers assigned sequentially, left to right, top to bottom. This applies to all image formats: screens, reports, etc. In the above example, the window names are PERSON.1, PERSON.2, PERSON.3, PERSON.4 and PERSON.5. There is a DataFlex command "NAME" which allows you to assign other symbolic names to each window.

If the /PERSON page above were displayed with the "PAGE" command or printed with the "OUTPUT" command, it would look just the way you see it above except that the appropriate data would have been MOVED into the data windows.

DataFlex will interpret PERSON as a page with 6 lines and 5 data windows. The type and length of each data window are as follows:

```
PERSON.1 - 10-character ASCII
PERSON.2 - 20-character ASCII
PERSON.3 - 3-character integer (no decimal) number
PERSON.4 - 4-character number, 1 place after the decimal point
PERSON.5 - Date
```

When the data windows are filled in, PERSON might look like this:

```
                 MY FIRST SCREEN IMAGE

      FIRST NAME: John         LAST NAME: Doe

      AGE:  25  WEIGHT: 156.5  DATE OF BIRTH: 12/12/1956
```

Note that the data is moved into the data windows by DataFlex COMMANDS.

If a configuration required two pages to accommodate additional information, the image format might look like this:

```
/PERSON

                 MY FIRST SCREEN IMAGE

   FIRST NAME: __________  LAST NAME: __________________

      AGE: __.  WEIGHT: ___._  DATE OF BIRTH: __/__/____

/JOB

                      MY PERSON'S JOB

      WORKS FOR: ______________________________________

        ADDRESS: ____________________________________

           CITY: ________________  ST: __  ZIP: _______

      JOB TITLE: ____________________
/*
```

This image has TWO pages: PERSON and JOB. Each page can be manipulated and addressed separately by name. The default window names for the second screen would be JOB.1, JOB.2 through JOB.6.

The IMAGE pages in a configuration are more than just a convenient way to format your input and output. They are actually part of the data structures of the configuration. Each data window can be used as a variable in which data is stored. You can read the data out of a data window as well as write to it.

When images are used for data entry it is unnecessary to input your data and MOVE it to a temporary variable. You may use the data window itself as the variable after you have given the command to accept user input to it.

For example, if you had previously ACCEPTed data into the window PERSON.3 (age) you could then use PERSON.3 anywhere that a numeric argument is acceptable.

EXAMPLE:

```
        MOVE (PERSON.3+1) TO PERSON.3
```

In data entry, any data that exists in a data window will become the default value for the record (existing or new) and can be edited or cleared with the DataFlex field edit keys defined in SETSCREEn.

WINDOW FORMATTING CHARACTERS

There are other window formatting characters that do not affect the data TYPE (ASCII, numeric, date) of the window but do affect the FORMAT of the data in the window. These formatting characters are used by placing them directly in the data window in the screen image. For example, a "," in a window specifies that numbers 1,000 and over should have commas inserted. These formatting characters only apply to output using the PRINT command. (For example, commas in numbers are not allowed in data entry.) The exception to this rule is CAPSLOCK, which works in both data entry and in print. If placed at the beginning of a window, these characters will not be interpreted; they will simply be displayed as though part of a prompt. They same is true if they are placed at the end of a window, with the exception of the decimal point, which in that position signifies a number with zero decimal places. The formatting characters must be embedded in (surrounded by) the data window character in order to function properly.

CHAR- ACTER	DESCRIPTION	FORMAT EXAMPLE	OUTPUT EXAMPLE
,	Embed commas in numbers	__,___.__	23456.99 = 23,456.99
C	Upper case only (caps lock)	_______C_	abc = ABC
0	Fill field to left with zero	___0____.	124 = 000000124
Z	Display zero as blank	____Z_.__	0.00 = (spaces)
@	Trim leading & trailing blanks	_______@_	ENTRYb̸b̸b̸b̸ = ENTRY
$	Floating $ on left	___$__.__	567.17 = $567.17

Calculations and string manipulations on data in formatted windows should be avoided, since formatting can change the apparent "value" of the contents for purposes of the calculation or manipulation. Calculations and manipulations should be done on the source data (database element, literal string, in-memory variable, etc.) from which the window was filled to begin with, with replacement of the results to the window, if desired for output purposes.

IMAGES IN DATAFLEX CONFIGURATION FILES

DataFlex configuration source files can contain IMAGEs and configuration commands. The configuration commands are action VERBS that interact with the operator, the IMAGE, and the data base manager. The data base manager handles the duties of storing, finding, and relating data stored on the system.

The configuration file has two parts: the IMAGE format section, and the commands section. The IMAGE section defines the interface to DataFlex. The commands specify what action will be taken by the configuration.

The first part of the configuration file is the IMAGE. The image section MUST start with a "/" in the first column of the first line. If there are no images for the configuration the first line will simply be "/*", which signals the end of the image section. If there are image PAGES for the configuration, the name of the first page will be on the first line and ALL of the images follow directly. The end of the IMAGE section is indicated by a "/*" on the last line of the image section.

The option "RESIDENT" can be placed on the same line as the page name. The RESIDENT option tells DataFlex to hold that page in memory at all times during execution of the configuration. This improves throughput speed when a page is used repetitively for output. Since RESIDENT does use up memory it should be used sparingly. Where the RESIDENT option is not used the image format will be called off of disk when it is needed. It takes less than a second to read an image from disk.

The "HELP" option can also be included on the page name line, marking the page as a help screen callable by pressing the HELP Flex-Key. See the section of "Using Function Keys" for further information on providing and accessing HELP screens.

After the "/*", the image section of the configuration ends and the command section starts. The command section consists of the commands and options listed in the command section of the manual. Commands exist for Data Entry, reporting, data base manipulation, PAGE manipulation, data movement, reading and writing sequential files, conditional processing and almost any other function you may require.

THIS PAGE INTENTIONALLY LEFT BLANK

DESIGNING DATABASES

ORGANIZATION OF DATA

Information, or DATA, can be organized in two ways: structured and unstructured. Unstructured data has no particular size or length or position. Examples of this would be: letters, books and the "pile of papers" on top of your desk. In a computer, word processors are usually used for processing unstructured data. When data is structured, groups of data are predefined and identified by what they contain. Structuring data allows for more consistent and efficient storage, retrieval and evaluation of the data. Business forms, such as an employment form, are an example of structured data that you may already use in a manual system. Accounting procedures are a method of structuring the financial data of a business. DataFlex is a tool for structuring ANY type of data in a computer system.

All of the data which makes up a particular information system is referred to as the "data base".

Groups of structured information about like things make up DATA FILES. Data files are analogous to a file cabinet drawer full of a particular business form. For example: a file cabinet drawer full of employee applications could become a data file of employee applications on a computer. DataFlex can have from one to 125 data files of structured information. Each is immediately accessible or "ON LINE" to the system. Each file is assigned a DataFlex FILE NAME and a FILE NUMBER to identify it uniquely.

Data files are made up of individual RECORDS. Each record would be analogous to ONE employee application in our example. A file is a group of like records. DataFlex can have from one to 64,000 records in each file. Each DataFlex record is automatically assigned a unique RECORD NUMBER when it is created. This number can be utilized as a system-assigned account number and is also useful when relating records from different data files together (as we shall see later).

Each record is composed of data "FIELDS". A field is one discrete piece of data that is of a defined type and size which is stored in a record. There are usually many fields in each record. Each field in DataFlex is given a name that is unique for that record. A record's number (see preceding paragraph) is automatically assigned in each DataFlex record. It can be used and displayed as if it were an ordinary numeric field. You cannot, however, change the record number or assign a record number yourself.

Data fields are sometimes referred to as ELEMENTS in the record. Examples of data fields in our employee record would be: Last Name, Date, Age, Job Sought, etc. DataFlex can have from one to 255 fields in each record. Each field is identified by an ELEMENT NAME.

In summary, a group of FIELDS is contained in a RECORD and multiple records of like kind make up a FILE.

DATA TYPES

In DataFlex, each data field can be assigned as one of three types. Each type uses the computer's storage differently. The basic storage unit of a computer is called a BYTE. You should know how many BYTES of storage your computer disk system has available. The three DataFlex data types are:

ASCII - This refers to the full set of printable characters that your computer can generate. It includes all letters, numbers and special characters. Examples of fields which would be "typed" as ASCII data include; names, descriptions, and notes. Each ASCII field is assigned a maximum length (number of characters) that the data in the field can occupy. Each ASCII character requires ONE byte of storage. Numbers (0...9) entered in fields that are of the ASCII type will be treated only as characters without numeric properties.

NUMERIC - The only characters that can be put into a numeric field are the numbers (0...9), a minus sign "-", and the decimal point ".". Numeric fields are used to store numbers. Examples are: prices, amounts, and quantities. Numeric fields can be used in calculations. In DataFlex, each TWO numeric characters defined in a numeric data field require ONE byte of storage, so numeric fields take up half the amount of space as ASCII fields. Numeric fields must be assigned a number of characters before and after the decimal point. This storage CANNOT be split over the decimal point, thus 1.2 and 10.2 and 10.12 would all require TWO bytes of storage.

DATE - Although dates can be represented as ASCII or NUMERIC fields it is more convenient to have a special data type for them. In DataFlex, a date can be entered in a date screen window in the following formats: MMDDYY, MM/DD/YY, or using any punctuation mark as a delimiter (e.g., MM.DD.YY or MM,DD,YY). Once the date data is entered it will be displayed formatted as MM/DD/YY. For applications involving dates outside the range 1901-1999, a four-digit year may be specified (MM/DD/YYYY) in the screen image. If this election is made, the format must be used consistently for dates throughout the application, and 2-digit year entries should be trapped out in the configuration, as they generate different internal values (e.g., 12/24/44 has a different internal value from 12/24/1944). In 4-digit mode, DataFlex can handle dates from the year 1 A. D. to the year 2500. The date is stored internally as a six-digit number representing the number of days since day one of year zero (01/01/00). These are called Julian dates. Julian dates can be used in calculations just like numbers. Thus you can add 31 to 12/05/81 and get 01/05/82. Date fields always require three bytes of storage.

It is important that each data field is assigned the correct type and is of sufficient length to hold the largest or longest possible value that you would want to put into the field. It may be worthwhile to abbreviate some long values to shorten the field length and take up less space in your record. The total storage requirement for each record, or the RECORD LENGTH, can be computed by adding up the length in bytes for each field in the data record. Space consumption is determined by a field's defined length, not the length of data actually entered into it from one record to the next.

Indexing in Applications

DataFlex has the capacity to store up to 64 thousand records in each data file. Therefore, it is important to be able to locate records of interest quickly and easily. For example, in a file drawer of employee file folders, you would probably have each employee's file filed in order by his name. That way you can "pull" the record from the file quickly, so long as you have the name.

Placing the records themselves in alphabetical sequence in a file is one way of organizing the data so that it can be found when needed. The method works whether the file is stored in a cabinet or on a computer system's disk. "Finding" something in an organized scheme is easy and efficient.

Consider however, that you wanted to find an employee who was interviewed a month ago (you know... what's-his-name). You know when he was interviewed, but not his name. To go directly to his record, you would have to previously have cross-indexed the file by date. Or, you could search the whole file until the desired record was found.

A computer has exactly the same options to find information in its files. It can search them, or maintain some kind of organized listing (index) of one or more elements of each record separate from the data file itself. For example, a file of employment applications could be indexed both by applicants' NAMEs, and by DATE RECEIVED. That way, the desired element can be searched for efficiently even if its order or sequence is different from that of the records themselves.

Even though a computer can scan its files MUCH faster than you could look through a card file, that process takes far too long to be practical each time that you want to review or change a record.

A telephone directory is a perfect example of how an indexing system works. The phone book is an alphabetical listing (an index) of the names of people with phones in a certain geographical area. Listed with the name is a number which can be used to "find" a person. The phone number in the telephone system compares functionally with the record number on a computer. To find a person, you "look him up" in the list. Then you "access" the person by his number.

DATAFLEX stores each new record at the end of the data file; figuratively, at the bottom of the existing list. It has, by virtue of its relative position in the file, a record number: first, second, third, etc. But that does you no good! To provide for rapid access to the records in a file, DATAFLEX keeps "indexes" of data that can be used to find a particular record. Each index is an organized list made up of like data elements from each record, and each record's number.

A computer index works the same as the phone book. It is an organized listing of selected data from each record in a file which includes the record number (like the phone number) of the record. Each record in a DataFlex system can be accessed or "found" by up to four different indexes(in 8-bit computers--16-bit computers support nine indexes). The designer of a DataFlex database can define which elements of each record are to be used to create the index(es), and thereafter, find the records.

DataFlex uses a B+ ISAM (ISAM stands for Indexed Sequential Access Method). B+ ISAM is the best method for computer indexing management available. DataFlex keeps index data files up to four ways simultaneously on 8-bit computers (nine ways on 16-bit). That means our employee file could be indexed by NAME and DATE and two other things (perhaps job classification and social security number), all at once. For speed, the DataFlex B+ ISAM keeps all indexes current **after each record entry**, without a long "SORT" process every time you enter data into a file. These features are referred to as MULTI-KEY, ON-LINE ISAM.

The creation and automatic on-line maintenance of multiple indexes is an important feature of DataFlex. This cross-indexing allows super-flexible access to any record in your data base in minimum time. Records can be located by entire or partial key. The database can be searched randomly or sequentially, by any index, without time-consuming reorganization or other clumsy procedures.

In addition to the KEY FIELDS discussed above, records can also be found by their RECORD NUMBERS just as rapidly as by an index. The system automatically assigns the RECORD NUMBER to each record when it is created. The record number of any record is always field number zero in the file definition.

Each of the indexes for a file can be made up of as many as 4 SEGMENTS. Think of a file of purchase orders. Each drawer is dedicated to one vendor, so VENDOR ID would be the primary indexing field. Within each vendor (drawer), there would be file folders--one for each part supplied by the vendor. The folders would be arranged in order by PART NUMBER. PART NUMBER would be the secondary indexing field. Within the folder for each part would be the orders for the part, filed by ORDER NUMBER, which would be the third (and final) indexing field. These successive levels of detail serve to identify each record (order) uniquely, and is an example of a three-segment index, consuming one of the 4 indexes available for each file in 8-bit environments. With the file drawers, the orders can be arranged only the one way, but with DataFlex, there remain three additional orderings (indexes) available (eight in 16-bit environments). An additional index could be another multi-segment index, say PART NUMBER/VENDOR ID/ORDER, for those situations where you would want vendors and orders grouped by part number. And there could be yet further indexes, multi-segment or single.

DataFlex will NOT allow two records in the same file to have exactly the same key value. However, it can appear to the operator that duplicate values have been accepted. To allow for duplicate records, the record number (designated as field zero in the file definition) is assigned as the last field segment in the index. Since record numbers are NEVER the same, the full index entry can never be duplicated (DOE,J/200 differs from DOE,J/75).

AUTODEF PROGRAM

The DataFlex utility program AUTODEF is the easy way to produce an ENTRY configuration from, and for, a freshly-drawn entry screen image. It is recommended for use even if your ultimate goal is beyond the actual capabilities of AUTODEF (in which case, after running AUTODEF, you would edit the resultant configuration source file with EDITOR or your word processor, and also edit the file definition with FILEDEF if necessary). It is also recommended for use in multi-entry-screen configurations to produce ENTRY configurations which can be appended together into the command section of the eventual ENTRY configuration, with the related screen images also appended together in the same order at the beginning of the same file.

AUTODEF's job is to read an image file (see Formatting With Images) which you have previously created either with the DataFlex Editor, or with any word processor of your choosing which is capable of producing ASCII files (most can). The image represents the input "form" that you intend the operator to see when entering to, or searching in, the database. With it, AUTODEF creates the data, index, and definition files, and the fields with their names and the type of data each field will contain.

rootname.FD	**File definition for the macro processor**
rootname.TAG	**File/field names**
rootname.DAT	**The data file**
rootname.Kn	**The index files, n=Index number**
rootname.FRM	**The source file**

To run AUTODEF, type AUTODEF at the command level and press the return key. Or, you may select the AUTODEF option from the DataFlex Configuration Menu. If AUTODEF is invoked with no argument (rootname of the database you wish to define) it will prompt you for a "ROOT NAME" as follows:

```
ENTER <RETURN> TO EXIT OR
"ROOT NAME" OF FILE DEFINITION TO CREATE:
```

The ROOT NAME is the name and drive specification of the file to be input to AUTODEF. For example, the following could be entered as ROOT NAMES: B:MYIMAGE or C:CLIENT or, where the input file resides on the logged drive, AN IMAGE. You may enter a command line of: AUTODEF ROOTNAME at the operating system level, in which case AUTODEF will skip the prompt for rootname.

After the rootname has been entered, (or if AUTODEF was invoked **with** an argument), it will ask the following question:

```
ENTER THE MAXIMUM NUMBER OF RECORDS "rootname" COULD HAVE?
```

You should respond with your best reasonable estimate, realizing two things:

a. If you elect too few records, it is always possible to go back to FILEDEF and increase the number of records in this database's capacity (after doing which, it will be necessary to run REINDEX on the database).

b. If your system option in SETSCREEn was set YES for pre-allocation of disk space required for databases, then AUTODEF will next go to your disk drive and "fill up" all the files necessary for this new database (indexes, data, and so on) with spaces to pre-empt enough space to contain the database when it is entirely full of data. Some multi-user systems require this, and it is advisable under other conditions, but you should ensure that you have sufficient free disk space available before entering a large number in response to this question when the system option is set to YES.

Naming the Fields

AUTODEF reads in your image and prompts for the assignment of a name for each image window. These names are assigned to identify the fields in the database file that has been created. When referring to these named fields in configurations, the correct format to use is **filename.fieldname**, where fieldname is the name you assign in this process, and filename is the name of the database file. You will be prompted for the names from top left to bottom right of the image. As AUTODEF proceeds through the fields, the underscores for the field to be named will be replaced with asterisks in the displayed image to show you which window you are currently being asked to name. Type the name you wish to assign to the field in the indicated window. You may use the BACK WINDOW Flex-Key to change a name assignment.

The names created here will be used throughout DataFlex to identify the elements of the database in the filename.fieldname format established above. The use of the element names MUST be consistent, and conform to the symbol name conventions established for all DataFlex data element references. They must start with a letter, contain only letters, numbers, "_", or "#". Fieldnames may not be longer than 15 characters.

NOTE: Your screen image must begin with the page name (up to ten characters, of which the first must be a letter, no spaces or punctuation, preceded by a diagonal (/). If this is omitted, AUTODEF will abort, displaying the message "ERROR, SCREEN IMAGE MUST BE NAMED. AUTODEF ABORTING!"

After all the fields are named, the bottom line prompt will ask:

```
Press "C" to continue or "R" to re-do this page _
```

At this point, you may re-do the page if you find that it has any errors by responding with an "R". Otherwise, answer with a "C" and continue to the next step.

The screen will now display the field names you have declared in the previous step, and prompt for which fields are to be used for indexing.

ENTER FIELD NUMBER TO INDEX OR <RETURN> TO END:

You should enter the number of the field(s) by which you want to FIND data in the file being created. Entering a field number, and then pressing <RETURN> will create an index containing data from the field whose number was entered. Multiple field numbers can be entered (up to 4 on 8 bit systems, up to 6 on 16 bit systems) to create indexes composed of multiple data fields. See the first part of this chapter for further information on indexing.

As field numbers are selected for indexing, the bottom prompt line will display:

WILL THE DATA IN THIS INDEX BE UNIQUE? <N>_

The default is N, which if selected will cause the record number to be combined with the data in the field itself to establish the index. This prompting will repeat until you have reached the maximum of 10 indexes for 16-bit systems, 4 indexes for 8-bit systems, or you key **<RETURN>** to end. As you select each field to index, the display will change to include an "*" between the number and the field name. If you wish to revise an index after creating it, pressing the BACK WINDOW Flex-Key will move execution back one field so you can re-define the index. Press <RETURN> without a field number entry when you have made all index designations you desire.

Upon the completion of assigning indexes, AUTODEF will create the files required for the configuration. AUTODEF will display momentarily on the screen several system messages and automatically return to the menu. The configuration file output of AUTODEF is called **rootname.FRM**. The **rootname.FRM** configuration source file may be modified using the EDITOR or your own word processor prior to compilation, or compiled "as is" when output from AUTODEF.

Rootname.FRM must be compiled to produce an executable DataFlex program. To execute a compile at the operating system command level, type COMP **rootname.FRM**. If you are operating from the DataFlex Menu, select the Configuration Menu and enter the option for "Compile a Configuration". See the chapter on the DataFlex Compiler for more information.

THIS PAGE INTENTIONALLY LEFT BLANK

HOW TO WRITE A CONFIGURATION

DataFlex configurations are semi-compiled. This means that the "source" file which you create must be translated by a separate step into a form which the computer can execute. Execution of a (semi-) compiled configuration then is actually faster than execution of an uncompiled source list would be. This section of the user's manual refers only to the physical process of creating the application. Other parts of the manual will instruct you on what commands and functions can be used within the configurations.

A configuration file is a set of compiled DataFlex commands which are ready to be used. A configuration is different from a program in two respects:

1. It can only be executed by the DataFlex runtime system.
2. Information from other systems, such as file definitions in the DBMS, affect the operation of the configuration.

The configuration determines the ACTION of your application in relation to the DEFINED file definitions and forms IMAGE definitions. Before a configuration can be used it must be:

1. Created in source form with the DataFlex EDITOR utility or any other text editor that produces ASCII files.
2. Compiled from the "source" file form into a file that is executable by the DataFlex runtime system (a .FLX file).
3. Run with the DataFlex runtime system in the presence of the other files required for execution (FILELIST.CFG and appropriate data and key files).

CREATING THE SOURCE FILE:

There are five ways to create the original source file for a configuration, of which four are "automatic" and the fifth, which is "manual," may also be used for editing configurations created by any of the other means. First, there are the AUTODEF and FILEDEF utilities with the "read screen image" option to create both the file definition and configuration source code for a data entry screen. Secondly, there is the QUERY utility to generate a REPORT configuration source code file. Finally, there is the READ utility, which creates configurations to convert ASCII data files to DataFlex data files. Lastly, there is the DataFlex EDITOR or any other text editor which is used to create the file from scratch. Modifications to any source code have no effect, of course, until the modified source code file is recompiled.

The configuration source files are standard ASCII files, so they can be listed and modified by means customarily employed for such files.

The format of a configuration file is quite simple: the IMAGE page(s) is (are) at the top of the configuration, each identified with a slash "/" in the first column of the first line followed by the name of the IMAGE page. The image format itself starts on the next line. An image is terminated with the next occurrence of a "/" in the first space of a line. "/pagename" marks the next image page, or a "/*" signals the end of the last image in the configuration. The first line of the configuration file must contain either a "/pagename" or a "/*", the latter indicating that there is no image in the file.

After the "/*" in the source file, the configuration commands start. These can be any of the commands listed in the command sections of this manual, or any commands you may have created with the macro system. Commands and filenames can be in upper or lower case-- DataFlex does not distinguish between the two except in screen images and literal (quote-enclosed) material. Command lines can be up to 255 characters long. If a command line reaches the end of a line on your terminal's screen, you can continue the command line on the next terminal line either by typing a semi-colon (;) followed by a carriage return, or merely let the line "wrap" on your screen.

The following is a sample of a simple configuration to display, and then allow the operator to modify, the system date:

```
------------------------------------------------------------
/DATEX


              TODAY'S DATE IS: __/__/__

Please enter today's date in MM/DD/YY format and press RETURN.
/*
OPEN SYSFILE                      // Open the system file
DISPLAY SYSFILE.DATE TO DATEX.1   // Show the date that is set
ACCEPT DATEX.1 TO SYSFILE.DATE    // Accept the updated date
SAVE SYSFILE                      // Save the system file
ABORT                             // Stop and end configuration
------------------------------------------------------------
```

HOW TO COMPILE A CONFIGURATION:

Once a configuration source file has been created, there is one more step which must be executed before it can be used. A compiled configuration must be created which the DataFlex runtime system can execute. This done by a program called "COMP.COM" (compile), or COM.CMD in CP/M-86 versions. COMPile reads the source file you previously created and produces a new, additional file with the special extension ".FLX" after its filename. The ".FLX" file can be executed directly by the DataFlex runtime system (see the section on "Operational Orientation" for further information).

This process is started by typing the word "COMP" followed by the full name of the source file to be compiled. A listing of the file as it compiles will be shown on the screen.

EXAMPLE:

```
        COMP datex.src
```

This will read and compile the source file DATEX.SRC and produce the configuration file "DATEX.FLX". The ".FLX" file is ready to run. This step must be done to produce a configuration that will run. The ".SRC" extension on the source file name is not required by the compiler, nor by DataFlex, but is recommended practice where extensions are not purposely being used for some other end.

On the screen listing of the configuration during compilation you will note some numbers and a ">" before each line. This is the line number of the command line actually output to the runtime system. If an error occurs during runtime this number will be shown with the error message to help you find the error. A command line in the source file can generate zero, one or several lines in compilation, so the line numbers output by the compiler may seem to skip over some line numbers as they would be expected to appear based on the source version. This lets you know how many lines a particular source command takes up. Most commands generate one output line.

Following is a complete list of the compiler (COMP.COM) command line options that can be used individually or in combination as below:

OPTION	FUNCTION
E	Stop/Pause on Error
S	Do not delete temporary files
F	Save output listing in file (.PRN)
L	Echo output listing on printer (LST:)
Dx	Save .FLX file on Drive x:
M#	Expand/contract memory for macro expansions

Compile Option E: Stop/Pause on Error Use of this option will cause compilation to stop if an error is encountered in compilation. The screen display will freeze, displaying the code which triggered the error, and some of the code above it, and the message to press any key to continue compiling. Once a key has been pressed, the option to abort the compilation (Yes or No) is offered.

Compile Option S: Do Not Delete Temporary Files Compilation produces two files every time it is run, which are automatically deleted when compilation finishes or is aborted. One file has the literal name of FORMTAG.TMP, and contains the names of the windows used by the configuration begin compiled, and the other has the variable name of filename.IC, and contains the Intermediate Code of the configuration (further explained in the section on "Modifying and Extending DataFlex). These files are not of interest to the general user, but if there is a reason why you wish them not to be deleted, this option will retain them.

Compile Option F: Save Output Listing in File The compilation process involves expansion of some commands into groups of sub-commands, and generation of line numbers for each line of this code. The numbered lines with sub-commands can be output to a file of the name filename.PRN by use of the F option. This option would be of use primarily to a user who has created custom macros and wishes to see their expansions in the context of the configuration they work it.

Compile Option L: Echo Output Listing on Printer The output described above under Option F can alternatively be printed on the printer during compilation. This is helpful for locating errors reporting during compilation, and is recommended for compiling more complex configurations.

Compile Option D: Place .FLX File on Drive x: If you wish the compiled configuration to be saved to a logical drive other than the one on which the compiler and source code are, the D Option, followed by the letter of the desired drive will cause the file filename.FLX to be written to the specified drive.

Compile Option M: Expand/Contract Memory for Macro Expansions When compilation begins, 4 kilobytes of your computer's memory is allocated for expansion of macro commands into subcommands for the compiled configuration (6K on 16-bit systems), and the remainder is available for the storage of symbols, which include pagenames, window names, database elements, in-memory variables and labels. Some configurations need more space for symbols than is available for them, producing the error message OUT OF MEMORY. When this happens, compilation can be reattempted with the option M#, where # represents the number of kilobytes of RAM to be made available for macro expansion from 1 to 9. In such cases, # less than 4 would be appropriate for 8-bit systems, and less than 6 for 16-bit systems. Conversely, in those cases where macro expansion space is inadequate (compiler error MACRO BUFFER SIZE EXCEEDED), the M option can be used to expand the available space as high as 9 kilobytes.

EXAMPLE:

```
        COMP myfile.frm ;ESFDB
```

Pause on error, save temporary files, create a .PRN listing file, and output MYFILE.FLX to drive B:

The semi-colon is part of the option syntax, and must be typed when options are used.

HOW TO RUN A CONFIGURATION:

Once the configuration has been created and compiled it is ready to execute with any installed DataFlex runtime system. At the operating system command level you simply type the word "FLEX" followed by a space and the name of the configuration. If "FLEX" is entered without a following configuration file name, the menu system (if present on the disk drive) will execute. As such, "FLEX" is the usual opening statement operators are taught to enter to initialize the system.

EXAMPLE:

```
FLEX DATEX
```

Executes the DATEX configuration shown above.

Once execution is under control of the DataFlex runtime, you never have to come back to the operating system, since one configuration can RUN another using the CHAIN command to completely automate your application as is done in the MENU system.

THIS PAGE INTENTIONALLY LEFT BLANK

EDITOR

The DataFlex EDITOR program is a stand-alone text editor for the creation and manipulation of DataFlex configuration files and images. It is invoked by keying:

```
        EDITOR <filename.ext>
```

from the operating system, where <filename.ext> is the name of the file which it is desired to edit or create. If the filename is omitted, EDITOR will initialize with a default filename of WORK, which may be changed upon saving to disk or, if it is expected to re-edit the file many times in succession, it may be saved under this name and re-edited by default each time EDITOR is invoked (with no filename).

While the primary purpose of EDITOR is to create the source files for DataFlex, it is also useful for general sequential file maintenance and limited word processing. EDITOR features full screen cursor movement, vertical and horizontal (up to 158 columns) scrolling, and insertion and deletion of characters and lines. There is also a search feature to search for patterns in text. ASCII text files can either be created or edited (changed) using EDITOR.

EDITOR uses the cursor to show where the editing process is taking place. The cursor's movement is controlled by the "FLEX-KEYs" assigned in the DataFlex SETSCREEN program. Some of the FLEX-KEYs keys perform the same functions in EDITOR as they do in normal DataFlex operation, while other FLEX-KEYS perform different functions from the ones they perform in DataFlex. This is because there are certain functions in EDITOR which are purely text-editing functions, and which are not supported in DataFlex operations. Finally, a number of the FLEX-KEYs are not used at all in EDITOR.

EDITOR FUNCTIONS ARE EXECUTED BY "FLEX-KEYS"

This section lists the functions of the DataFlex EDITOR followed by the FLEX-KEY which executes the EDITOR command.

RETURN: (RETURN KEY)

The RETURN key will enter text on a line and advance the cursor to the next line of the display. If the cursor is on the last line of the display, hitting RETURN will cause the displayed text to scroll up one line. If the cursor happens to be in column 1 (one) of the text, then hitting the RETURN key will cause an insertion of a blank line at the point where the cursor was.

UP ARROW: (UP ARROW KEY)

The UP ARROW key will move the cursor up one line on the screen. If the cursor is on the first line of the display then the text will scroll up one line.

DOWN ARROW: (DOWN ARROW KEY)

The DOWN ARROW key will move the cursor down one line on the screen. The screen will scroll down one line if necessary.

RIGHT ARROW: (RIGHT ARROW KEY)

The RIGHT ARROW key will move the cursor one character to the right without erasing any existing characters.

If the cursor is at the right boundary of the screen the display will scroll horizontally to the right. The maximum length of a text line is 158 characters. The cursor will not move into unwritten space.

LEFT ARROW: (LEFT ARROW KEY)

The LEFT ARROW key will move the cursor one space to the left without erasing any existing characters. The cursor will not move past the left boundary of the screen unless the screen has HORIZONTAL SCROLLED to the right by a previous RIGHT ARROW command.

DELETE END OF LINE: (DESTRUCTIVE BACKSPACE KEY)

The DESTRUCTIVE BACKSPACE key will move the cursor one space to the left, deleting all the text from the cursor to the end of the line.

INSERT CHARACTER: (INSERT CHARACTER KEY)

The INSERT CHARACTER key will move the text line to the right by one space opening up the text for the insertion of a character at the cursor position. If the line contains the maximum number of characters (158) then the last character will be lost. The key must be pressed one time for each character to be inserted.

DELETE CHARACTER: (DELETE CHARACTER KEY)

The DELETE CHARACTER key will delete the character under the cursor and move all text to the right of the cursor one space to the left.

INSERT LINE: (SAVE RECORD KEY)

The SAVE RECORD key will insert a line above the line on which the cursor appears. After this key is pressed, the screen will scroll all text below the cursor, and leave a blank line to edit.

DELETE LINE: (DELETE RECORD KEY)

The DELETE RECORD key will delete the line on which the cursor appears.

LOAD FILE: (BACK FIELD KEY)

The BACK FIELD key will <u>clear the text buffer</u> and ask for the filename of the text file to load. <u>If this key was pressed by mistake, press ESC</u> to abort and return to the text in the buffer without loading new text.

SAVE FILE: (USER-DEFINED FUNCTION KEY)

The USER-DEFINED FUNCTION key will save the text file in the buffer to the disk. If you change your mind after invoking this command, answer its query by pressing the ESCape key. The prompt will offer the opportunity to save the file under its original filename (WORK if EDITOR was invoked with no filename specification) by merely pressing the RETURN key. If a different filename is desired after an edit session, a new filename (with drive specification if necessary) may be entered in response to the prompt.

ESCAPE: (ESC KEY)

The ESCape key will exit the document editing portion of EDITOR and allow the naming of the output file to the disk. The output file should conform to your standard operating system file name conventions (generally a two character drive specification, eight characters for the file name plus a three character file type extension).

TAB: (FIND KEY)

The FIND key will tab eight spaces to the right or to the left depending on the last direction that the cursor was moved.

BEGINNING OF LINE or HOME: (CALCULATE KEY)

The CALCULATE key will move the cursor to the beginning of the line being edited. If the cursor is on the first character then the cursor will be placed at the top of the text.

BACK PAGE: (PREVIOUS RECORD KEY)

The PREVIOUS RECORD key will page the cursor backward 24 lines.

FORWARD PAGE: (NEXT RECORD KEY)

The NEXT RECORD key will page the cursor forward 24 lines.

FIND TEXT: (SUPERFIND KEY)

The SUPERFIND key will display the prompt "Find string:" at the bottom of the screen, to which the operator should respond by keying in the string which it is desired to find in the file. Once the string has been found, it may be edited, deleted, or whatever is desired. If the SUPERFIND key is pressed again, the prompt will reappear. If the key is pressed again, in response to the prompt, the string entered previously will be searched for again in the file, and will be searched for every time the SUPERFIND key is pressed twice, until a different string is entered in response to the prompt.

REINDEX UTILITY

The DataFlex index files are maintained dynamically during normal system operation. No reorganization is generally necessary. However, due to changes made in the file structure after the entry of data, adding new indexes or system failure, it may be necessary to rebuild or create new index files. The REINDEX utility provides this function. Each index is an independent file with a .Kx extension, where x is a number which distinguishes the index from other indexes of the database. REINDEX can rebuild all indexes, batch indexes only, (see next paragraph) or just one particular one.

There are TWO types of index files in DataFlex; ON-LINE indexes which are maintained dynamically during file creation and maintenance and BATCH indexes which are usually used for infrequently-used report sequencing. The batch indexes are NOT updated when a record is created or deleted as the on-line indexes are.

There is a 5th index (on 8 bit systems--10th index on 16 bit systems) that is reserved for "ad hoc" sorting of existing files. REINDEX can redefine the specifications for this index and create a new index interactively. Any other BATCH index can also be redefined from the command line.

WHEN TO USE REINDEX:

The following are indications that your index files may be damaged and you must run REINDEX: 1) A DataFlex error 20, 21, 22 or 26; 2) Records seeming to be "out of order" when an attempt is made to list them in order; 3) Trying to FIND one record and getting another; 4) Records "missing" out of the data file. For any of these indications it is recommended that the FREL program be used on the suspect data file before REINDEX in case the data file itself is damaged.

USING REINDEX:

REINDEX can be used in two ways: instructions can be given on the command line that calls REINDEX, or its operation can be controlled interactively. First we will deal with the interactive mode.

When REINDEX is called with no arguments on the command line, you will be greeted with a files menu. The active DataFlex files will be displayed with their file numbers. You should select the file you wish to REINDEX. If you have more than 32 active files you may press <RETURN> to get more files listed. At any time within REINDEX you may hit <ESCAPE> to return to the menu. (Note: the file selection process is the same as that in QUERY).

Once you have selected a file, the following menu will be displayed. You must select an operation according to your needs:

```
                              DATAFLEX
                           REINDEX UTILITY
                             <FILE NAME>


                      1..... RE-CREATE ALL INDEXES

                      2..... RE-CREATE ALL BATCH INDEXES

                      3..... RE-CREATE ONE INDEX

                      4..... DEFINE & CREATE "AD HOC" INDEX

                      5..... DELETE "AD HOC" INDEX


    PLEASE ENTER OPTION REQUIRED:
```

OPTION ONE - RE-CREATE ALL INDEXES

Re-create all indexes will read the data file selected and rebuild all defined indexes. This is generally used when a system or power failure makes the validity of the index file questionable.

Note that if the validity of your data file is in question, the FREL program should be used before the REINDEX program.

OPTION TWO - RE-CREATE ALL BATCH INDEXES

Re-create BATCH indexes will create all indexes marked as "batch" for the selected data file. This is generally used before generating reports that require the batch indexes, i.e., before end-of-month reporting.

OPTION THREE - RE-CREATE ONE INDEX

Re-create one index, allows you to remake indexes, one at a time. This may be required if you made a new index with the FILEDEF program or you had a disk error on an existing index. Option three will list the available indexes and allow you to select the index you wish to re-create.

OPTION FOUR - DEFINE & CREATE "AD HOC" INDEX

Option four allows you to create an "AD HOC" batch index. This index can then be used in any DataFlex output. This option will ALWAYS create index 5 (or index 10 on 16 bit systems). When selected, REINDEX will display a field list. You may then select up to four (4) fields to be included in the index (6 on 16 bit systems). If there are more fields in the selected file than will fit on the screen, you may use the <NEXT RECORD> key to get more fields or the <BACK-RECORD> key to start at field one again. Once you have selected the fields to index by field number, press <RETURN> and REINDEX will proceed to create the new index. (Note: this field selection process is the same as that used in QUERY).

OPTION FIVE - DELETE "AD HOC" INDEX

Delete ad hoc index, will remove an index file created in option four from the disk and the file definition. This will reclaim the disk space used by the index.

REINDEX - COMMAND LINE FORMAT

The use of REINDEX with the arguments on the command line is primarily intended for use by configurators within "submit" procedures. If you are happy with using REINDEX interactively you may skip this section.

All of the above options and more can be invoked in configurations with command lines:

REINDEX <FILE-NUMBER>
will re-create all indexes for (FILE). This is equivalent to Option One on the RE-INDEX Menu.

EXAMPLE:

```
        REINDEX 22 - Recreate all indexes for file 22.
```

REINDEX <FILE-NUMBER> MENU
Will select the file number and proceed to the REINDEX menu for user interaction.

EXAMPLE:

```
        REINDEX 22 MENU - Select file 22 and display REINDEX menu.
```

REINDEX <FILE-NUMBER> BATCH
Will recreate all batch indexes for <FILE>. This is the same as Option Two on the RE-INDEX Menu.

EXAMPLE:

```
        REINDEX 22 BATCH
        Recreate all batch indexes for file 22
```

REINDEX <FILE-NUMBER> <INDEX-NUMBER>
Will re-create the specified index for the specified file.

EXAMPLE:

```
        REINDEX 22 1
        Recreate index one of file 22.
```

REINDEX <FILE-NUMBER> <INDEX-NUMBER> <FIELD1> ... <FIELD4>
will redefine <INDEX-NUMBER> to the field numbers that follow. This is like Option Four, but any index can be re-defined. The index will be a BATCH type unless it was on-line before REINDEX was run.

EXAMPLE:

```
        REINDEX 22 5 4 5 0
        REINDEX file 22, index 5 by fields: 4, 5 and 0.
```

REINDEX <FILE-NUMBER> <INDEX-NUMBER> DELETE
Will delete the ad hoc index <INDEX-NUMBER>. This command works only on ad hoc indexes.

REINDEX <other options> QUIET
QUIET put at the end of the command line will cause REINDEX to display only a "*" on the screen for each record instead of the index data. Shortening the display increases throughput speed.

REINDEX <other options> .BAD
Will create a file by the name rootname.BAD with a list of any duplicate records found during reindex.

REINDEX <other options> FLEX <config-name>
Will CHAIN back to the Flex configuration <config-name> after the reindex is done. This gives you the capability to run a configuration (like a report) after a reindex operation.

EXAMPLE:

```
REINDEX 22 BATCH FLEX MYREPORT
```

DUPLICATE RECORDS IN REINDEX:

If REINDEX encounters duplicate records in any non-ad hoc index, it must delete one of the records to maintain the requirement that all index entries be unique. You will be given the option to: 1) NOT DELETE BAD RECORDS and leave the file in an improper state; or 2) List the bad records to a file or printer. The file of bad records will have the name rootname.BAD

FREE LIST UTILITY

The list of available (deleted) records in a file is referred to as the "FREE LIST". It is dynamically maintained during normal system operation (unless "REUSE DELETED SPACE" is set "Y"es in SETSCREEn). But due to system failure (disk errors or power fail) it may be necessary to rebuild the free list. This is done with the FREL utility. FREL will also check the file for type consistency (bad data) and delete any bad records.

WHEN TO USE FREL:

Conditions that would indicate the need for FREL and REINDEX are: Power failure with data files open. Disk errors, Errors 78, 81 or 1..6.

USING FREL:

FORMAT:

> **FREL**
> (to display a list of existing data files before executing FREL, or
>
> **FREL <FILE NUMBER>**
> (to execute FREL on a file whose number is already known.)

EXAMPLE:

> **FREL 10**
> Check file 10 for consistency and rebuild its free list

If running FREL indicates any bad records, you should rebuild the indexes for that file with REINDEX after running FREL.

FREL AND BAD RECORDS:

If any bad records are encountered during FREL, the record(s) will be deleted and the data from them can be sent to a file or to the printer. If the option ".BAD" is put on the command line, the bad records will always be listed to the file "rootname.BAD". When the first bad record is encountered, you will be given the option to: 1) Not delete the record and leave the file in an improper state; 2) List the bad records to the printer; or 3) list the bad records to the .BAD file. NOTE: Although there are many ways "bad" records can be created, a consistent attribute of bad records is the presence of control (non-ASCII) characters in data fields. FREL lists as "bad" any record having such characters in it.

THE READ UTILITY

DataFlex's READ program has the function of automatically writing configurations which "read" standard ASCII format data files and produce DataFlex data files, complete with record numbers and any indexes of the data which might be desired. It does this by querying the user as to the characteristics of the source file, and certain desired attributes of the DataFlex file to be created.

There are two main uses for READ-produced configurations: (a) to "import" data from non-DataFlex software environments such as MBASIC*, VisiCalc* DIF, dBASEII* SDF, SuperCalc* SDI, MailMerge*, and the like; and (b) to re-READ ASCII data from a predecessor DataFlex database into a newly-created DataFlex database having more or fewer fields than the original (or changes in their length or data type).

In order to provide a realistic context for the explanation of READ, the following instructions cover how to convert a data file from dBASEII* format to DataFlex. If your particular purposes do not involve dBASEII, merely skip those sections whose headings indicate that they pertain to dBASE, and substitute the appropriate instructions for your source environment. The overall process involves the use of DataFlex utilities other than READ. This interaction between READ and AUTODEF (and possibly other utilities) is, again, typical of most uses to which READ would likely be put in actual use.

HOW TO CONVERT DBASE II FILES TO DATAFLEX

Since this procedure involves producing two complete copies of the files to be transferred (the intermediate one may be erased after the process is complete), the transfer of large files may require some planning of disk space, and even of processing time for very large files. For this reason, it is recommended that the preparatory work be done with a subset of any large file which it is desired to convert, and that the results of that work then be applied to the entire file after its efficacy has been confirmed.

This procedure can be executed on small data files using a demo version of DataFlex. If you should encounter the error "DEMO LIMITATION EXCEEDED", it will be necessary to re-initialize DataFlex before you can continue your work from the last disk access prior to the occurrence of the error. Naturally, make sure your data file is backed up before you proceed. From dBASE, print out the database's structure for reference in the following steps.

MAKING THE MAIN ENTRY SCREEN

The first step is defining your database in DataFlex, a process which can be amazingly easy if you happened to develop the main entry screen for your dBASE database under dBASE's ZIP screen and report formatter (and retained at least your .ZIP files from the effort). What you need is a plain ASCII file showing the fields of your data elements, and their types.

If You Don't Have .ZIP Screen Files

If all you have is your original entry command file (.CMD) full of "@ l.c SAY caption GET variable" statements, then modify the .CMD file by SETting FORMAT TO PRINT right before the first "@" and run the program with your printer on. That will print out your screen, which you will then have to "draw" on the screen in ASCII, for which purpose your favorite word processor or DataFlex's EDITOR will do fine. Now you can go on to the final screen preparation steps below which are common to both manually and ZIP-created entry screens.

If You Kept .ZIP Files

ZIP in use puts out its own file with a .ZIP extension, a .CMD or .FMT file for dBASE which you won't need anymore, and a .ZPR file intended for use in printing out a copy of the screen. This .ZPR file is what you can use to create the screen image for DataFlex. If you erased your .ZPR file but saved the .ZIP file, you can recreate the .ZPR file just by loading the .ZIP file into ZIP and resaving. (Caution: if you use a .CMD or .FMT file produced by ZIP and then edited under its original name, resaving from ZIP will overwrite the file with its earlier ZIP-generated version. You can avoid this by renaming your output file when ZIP gives you the chance. But none of this really matters, because you're working on a backed-up disk, right? Right!) Load the .ZPR file into your favorite word processor, or the DataFlex EDITOR, and erase the first line, which reads "***** File filename *****". Then delete all embedded commands (those in [brackets]) and all #variable GET commands and @variable SAY commands, together with the variable names.

Final Preparation of the Screen

What you should now have, whether you got it from a .ZPR file or "drew" it manually, is a screen with its title, if any, and a prompt for each data item to be input in a logical position on the screen followed by enough space on the line for the data item to be entered into. Now, with this file loaded into either your word processor or DataFlex's EDITOR, you should draw a blank with underscores after each prompt in the same position as you used to have your GETs and SAYs. Each blank should be as long as the field length defined in dBASE (this is how DataFlex will figure out the field length). For numeric fields, there is one further step. Replace one underscore with a period wherever the decimal place should go in the field, remembering that that decimal point will actually provide space for the 1's position in your number (first place left of the decimal point). The blank for a five-digit integer (no decimals), for example, would look like: "____.", without the quotes, of course. That's right: four underscores and the decimal point. If any of the fields is a date, make it look like this: "__/__/__", remembering that DataFlex actually supports dates as a separate format, unlike dBASE. Last, open up a new line at the top of the screen (don't worry if the bottom line of your image scrolls off the bottom--this new line won't be part of the displayed image). At the leftmost position, give the screen a name for DataFlex to use, preceded by a diagonal (/ENTRY_SCREEN, for example). Save the file under a name which will be meaningful to you. Now your screen is ready for DataFlex to read it and create your new DataFlex database structure, all **from the** screen.

CREATING THE DATABASE STRUCTURE WITH AUTODEF

Now it's time to "feed" the screen image to AUTODEF. AUTODEF will define fields only if they're in the image. If you need fields that aren't in the image, you'll either need to add them to the image (they can be removed later), or edit them into the database using FILEDEF, but that's another story.

From the operating system, or the DataFlex menu, run AUTODEF (it's a .COM file). Answer the first prompt with the filename of the screen image file and tell it you're creating a new database, rather than editing an old one. Then tell it how many records you're going to need the database to hold (maximum 64,000). AUTODEF will then display your screen image and walk you through each of the blanks you provided (they're called "windows" in DataFlex), asking you what you want to name the variables. You can use your old dBASE names if you like, but DataFlex allows 12 characters in fieldnames, so you may wish to expand them somewhat to improve intelligibility.

Then AUTODEF will ask you which field(s) are indexed. Give it the same fields you used in dBASE. Be sure to give it at least one, so DataFlex will be able to find records, unless you are displaying the record number, in which case you don't have to have any indexed fields.

After this, AUTODEF will create a file (called a configuration file in DataFlex) identified as "filename.FRM" which will contain not only your screen image and a means of displaying it, but also a complete facility for searching the database by its indexes, editing, deleting, and adding records, **with no additional commands, embedded or otherwise.** Of course, if you do later wish to add bells and whistles to your entry configuration, you can edit the .FRM file with your word processor or the DataFlex EDITOR to provide them.

To use this configuration, you must compile it with the DataFlex compiler, selectable from the menu or executable from the operating system as "COMP filename.FRM". It will create the file "filename.FLX", which can be executed either from the DataFlex menu or from the operating system with the command, "FLEX filename". No point in running it yet, though. We haven't brought the data over for it yet. This is a good time to compile it, however, so from the operating system, issue the command "COMP filename", and DataFlex will do the rest over the next two minutes or so. Compilation is a step which must be taken with every DataFlex configuration you create, by whatever means, and a configuration must be re-compiled each time you want to make a change in the way it runs. This step isn't present in dBASE, so don't forget to do it in DataFlex when it's called for.

BRINGING HOME THE DATA

The last step is bringing the data over, something you should do first with a small test data file and then later with the whole file after you've made sure everything works right.

The dBASE Part:

Start dBASE and bring your database into USE. Do not invoke any dBASE indexes if you want DataFlex to end up with the same record numbers as dBASE was using (which wouldn't be of concern unless you use record numbers as account numbers or some such scheme). If you have deleted records in your database, you will have to manually re-delete them in DataFlex, unless you PACK your database first to eliminate the records. This, of course, will readjust your record numbers in dBASE. If you're converting a large database with one or more indexes, the conversion will run faster if you do use the largest index maintained in dBASE. This assumes that you will want the same index in your DataFlex version. The largest index is the one whose index file (.NDX) is larger than any of the other .NDX files. Your record numbers will change, of course.

Now COPY the database to an ASCII delimited file with the command:

```
        .COPY TO filename SDF DELIMITED
```

If some of your records have apostrophes (Jerry's Pool Hall) in them, you will want to make the command look like this to use double quotation marks instead of the apostrophes otherwise provided by dBASE:

```
        .COPY TO filename SDF DELIMITED WITH "
```

Make sure "filename" is the same filename as AUTODEF created from your image file. dBASE won't write directly into the DataFlex file, but it will create the file "filename.TXT", an intermediate version of your data. Also make sure your disk has space on it for a copy of your data file that is as big as the original. If your data file is large (over 100K) this process will take some time--hours for really large data files over a megabyte.

If you're running on a crowded hard disk or otherwise have less free disk space than your data file, there are options you can use in the COPY command which will enable you to COPY the file in pieces. Again, if you're utilizing an index, COPY the file out in the index sequence. For example, if your index is on last name, COPY can be run to first output all last names beginning with A through C, then D through F, and so on. This process can easily be put into dBASE and DataFlex command files/configurations that call each other until all records have been brought over, and run as a batch. After completion, check the number of records in the DataFlex file to make certain that no records were missed.

The DataFlex Part

The last step is done by DataFlex's READ utility, which is itself a DataFlex configuration, and whose operation creates a configuration which causes DataFlex to read your filename.TXT file and put data into your filename.DAT (DataFlex) file, complete with all indexes as specified in AUTODEF. Like any good DataFlex configuration, READ has lots of help in it, accessed by pressing the HELP key, so use it whenever questions come up. READ is invoked either from the DataFlex menu or from the operating system with the command "FLEX READ".

READ starts up with a screen that lists the DataFlex files already on your disk. You should see the one you created when running AUTODEF, with a selection number to its left. Respond with that number. Then a new screen will display the name of this target file, and ask you the name of the file to read **from.** This, of course, will be the file you created (filename.TXT) from dBASE's COPY command in the step above. Enter this name. Then READ will ask you the name you want to give to the configuration source file. What configuration source file? READ will convert your source data file using a DataFlex configuration which will actually be written by READ when you're done telling it all it needs to know about the desired conversion. So give it a name (I used DBASREAD), but **do not** give it the same name as you gave your image file (and resultant data entry configuration) in AUTODEF above, or your old file will be overwritten by the READ configuration, and that wouldn't be good.

After you've given it the name for the configuration file, the screen will clear, and READ will ask you whether your source file is line- or comma-delimited. If you did everything as recommended above, your file is comma-delimited, and you can tell READ this by entering the entire word "COMMA" in the window. The READ will ask you how many fields are in each record of your source file. Refer to your dBASE structure printout and count the fields for the answer to this question.

Next comes the part that could be tricky if you have made any change in the sequence of fields as they were in the dBASE structure on the one hand, and in the DataFlex windows in the screen image on the other hand. For each field of your source file, READ will ask you which field number (sequentially) of the DataFlex database it goes to. If your dBASE file structure and your DataFlex field sequence step along together exactly the same, this step is easy. One goes into 1, two into 2, three into 3, and so on. But if the first field of your dBASE structure is LASTNAME, and the second is FIRSTNAME, while your DataFlex screen image starts off with FIRSTNAME and has LASTNAME second, you'll have to tell READ about the switching at this point. Compare your printouts to make sure about any possible switches. If there are fields in your source file that you don't want transferred (or if there are fields in your DataFlex file that you don't want data transferred into), you can do that here by skipping source file fields with the <RETURN> key or skipping target file fields merely by not using their sequential numbers.

After this, READ knows all it needs to know, and will ask you if you want to compile the configuration immediately. Since you have to compile the configuration before you can use it, the usual response here would be "Y"es. The question is asked to give you the opportunity to edit the conversion configuration before compiling it, if you want to. Normally, you wouldn't want to.

READ will then write the conversion configuration and compile it. But READ hasn't done your conversion yet--it's just put everything in place for you to do it. Your final step is to to run your conversion configuration from the "Run a Configuration" choice on the menu, or give the command "FLEX filename", where filename is the name you gave the conversion configuration in READ. If you first run the configuration on a small test database and everything works all right, running it on the large, real database is as simple as doing the dBASE conversion steps on it, putting the resultant .TXT file in the right place where your READ configuration can find it, and giving the "FLEX filename" command again. You must have the same filenames as you used the first time, of course.

WHAT REALLY HAPPENED HERE?

In the foregoing, we did one conversion step in dBASE, creating one processed copy of your original data file with a .TXT extension. Either manually or from a .ZIP dBASE file, we created a screen image for DataFlex with any extension, which you might have named FLEXFILE. We then ran AUTODEF on that image file, producing a number of files, some of them empty data and index files, as listed below. Then we ran READ, generating a DataFlex configuration specifically to read your dBASE-generated file, convert its contents, and put them into the files created under AUTODEF. This is a list of the files in the sequence in which they would have been created:

	FLEXFILE.DBF	Your original dBASE data file
	FLEXFILE.TXT	COPYed version of the dBASE data file
	FLEXFILE.ZIP	File from ZIP for the dBASE entry screen
	FLEXFILE.ZPR	Printable version of the ZIP file
	FLEXFILE.	DataFlex screen image (possibly from .ZPR file)
	FLEXFILE.FRM	AUTODEF-created entry configuration source code
*	FLEXFILE.FLX	Compiled (executable) entry configuration
*	FLEXFILE.DAT	DataFlex data file created by AUTODEF
*	FLEXFILE.K1	DataFlex index #1, if any, created by AUTODEF
*	FLEXFILE.Kn	Additional DataFlex indexes, if any
*	FLEXFILE.TAG	Names of DataFlex fields, created by AUTODEF
*	FLEXFILE.FD	Definition of DataFlex database
	DBASREAD.FRM	READ-created source code for conversion config.
	DBASREAD.FLX	Compiled (executable) conversion configuration

The *asterisk indicates the minimum files which you must retain in order to use your new DataFlex database. Additional database and/or configuration files may, of course, be created. All the others may be retained for future reference or modification if desired (and space allows).

WHAT DO WE HAVE HERE?

The product of these efforts is much more than just a data file that DataFlex can use. In addition, we have created complete indexes for the file, to facilitate rapid finding of records in the database, including any we may have had in the dBASE version. Further, in AUTODEF, we have created a complete display and entry facility for the database records, through which we can query, edit, or add to, the new DataFlex database.

All further processing of the data and related configuration files to take advantage of DataFlex's many capabilities not present in dBASE (multiple commmand files open simultaneously, true multi-user operation, Julian date functions, and more), may be done on the DataFlex file family entirely within DataFlex. You're on your way!

DATAFLEX 1.6X TO 2.00 CONVERSION

The DataFlex conversion programs provide for the transfer of existing DataFlex 1.6X applications to the new 2.00 syntactical representation.

The input to the conversion process is DataFlex 1.6X ENTER and REPORT configuration files. The files must be working 1.6X DataFlex configurations. The output of the conversion process is DataFlex 2.00 source files. Before they can be run, the source files must be compiled into executable configuration files using the DataFlex compiler.

The DataFlex 2.0 compiler requires the creation of a new support file (FILENAME.FD) for each existing data (.DAT extension) file in your 1.6X application. This .FD file will be created by the 2.00 FILEDEF utility when a file definition is saved. The DataFlex FILENAME which is given to a data file will be used as the base name of the new file that will be created, and an extension of ".FD" will be appended to it. This file will be used by the conversion program and the compiler to assign symbolic names to the individual fields in the data file. These symbolic names will be composed of the DataFlex FILENAME and the FIELD NAME that each field has been assigned. Valid characters that can be used as part of a FIELD NAME are [A..Z, 0..9, #, @, _, $]. If other characters are used in FIELD NAME they will be automatically converted to an underline character by FILEDEF.

To create the .FD file, run the 2.00 FILEDEF program and select the data file that is to be converted. Once the file has been loaded, use option three (3) in FILEDEF to inspect the data file's DataFlex FILE NAME and all the FIELD NAMES in the file. If the DataFlex FILE NAME has not been filled in, or the FIELD NAMES are not correct, you may correct them at this time.

Once the modifications are complete, use option nine (9) in FILEDEF to save the data file definition (.DAT files) to disk. The 2.0 version of FILEDEF will at this point automatically create and write the needed .FD file. This process must be performed on all data files used by configurations which are to be converted to DataFlex 2.00. The .FD files must be on the same logged device as the compiler when the source configuration is compiled. Data files created under Versions 1.6X are otherwise completely compatible with DataFlex Version 2.0.

CVTFRM is used to convert the ENTER CONFIGURATIONS (.FRM) from DataFlex 1.6X to DataFlex 2.00 and CVTRPT is used to convert the REPORT CONFIGURATIONS (.RPT).

The conversion program requires only two file names to perform the conversion: the name of the version 1.6X configuration file, and a name for the Version 2.00 source file which is output by this process. The Version 2.0 output file name must be different from the Version 1.6X configuration file name.

EXAMPLE: A sample input filename is VENDOR.FRM, and the output filename will be VEN.FRM. Once the conversion is complete, it is necessary to compile the configuration into an executable file.

```
        A>CVTFRM
          ENTER SOURCE FILE NAME: VENDOR.FRM
          OUTPUT FILE NAME: VEN.FRM
               (conversion is done here)
          CONVERSION DONE.
        A>COMP VEN.FRM
```

ERRORS that may be encountered:

a) CANNOT FIND INPUT FILE
The conversion program can not locate the input file. Check the input name for spelling on disk directory to be sure it is there.

b) INPUT AND OUTPUT FILENAMES MUST BE DIFFERENT
The conversion program requires that input and output name be different. If an ouput file of the same name exists, it will be overwritten.

c) CANNOT FIND .FD FILE
A data file has been declared by the configuration program which is not found on the default drive.

d) FILE <<#>> IS NOT OPEN
A data file which has not been opened has been referenced by the configuration.

e) FILE <<#>>, FIELD <<#>> IS OUT OF RANGE
A field number has been referenced by the 1.6X definition configuration which does not appear in the data file.

f) WINDOW # <WNUMBER> OUT OF RANGE

g) UNSUPPORTED OPTION # ON LINE #
The original 1.6X configuration has an option number in it which is invalid (under both versions). The message shows the line number on which the invalid option number appears. The original configuration must be corrected before conversion is re-attempted.

h) OVERLAPPING IMAGE SECTIONS ARE NOT ALLOWED
Under some circumstances, DataFlex Version 1.6X Report section line declarations can include one or more line(s) in both of two adjacent sections. This condition is impossible for the conversion program to handle, and must be corrected in the 1.6X configuration by actual duplication of the common lines, and changing of the section line declarations so as not to include any of the same line numbers.

All of the errors above are fatal, and will abort the conversion process.

MENU CREATION AND MAINTENANCE

DataFlex is supplied with a powerful, versatile, and easy-to-use system for creating and maintaining menus which supports the execution of DataFlex configurations, other executable (.COM and .CMD) files, and prompting for the entry of necessary operator input at run time. The menu system can be made to execute upon cold boot, providing a "shell" which positively insulates the operator from the computer's operating system at all times.

This utility is itself a DataFlex configuration (filename MENUDEF.FLX), and it uses and creates data files for the menus under the df_filename "MENU". Like any good "Flex" application, MENUDEF is fully prompted, and menus can easily and rapidly be created or modified simply by following the prompts and answering them. MENUDEF also has extensive HELP screens, which can be called at any time during execution by pressing the HELP Flex-Key.

CREATING AND SELECTING MENUS

Upon selecting "Menu Definition" from the DataFlex Configuration menu, or invoking MENUDEF from the operating system with the command FLEX MENUDEF, a list of the numbers and headers of the menus currently on the disk will appear as shown below. The "headers" are the legends appearing at the top center above the menus, identifying what each one is:

```
DATAFLEX MENU SYSTEM                                      MENU SELECTION
========================================================================

  NUMBER                     HEADER

     __           ______________________________
                  ______________________________

     __           ______________________________
                  ______________________________

     __           ______________________________
                  ______________________________

     __           ______________________________
                  ______________________________

     __           ______________________________
                  ______________________________

ENTER NUMBER OF THE MENU YOU WISH TO EDIT: __
You may use the NEXT or PREVIOUS RECORD Command Key for more menus
or press <RETURN> to start a new menu.                 (HELP screen
                                                         available)
```

This screen will display all menus now existing, in numerical order. If there are more than five menus on the drive, pressing the NEXT RECord Flex-Key will display the next five menus. Pressing the PREVious RECord Flex-Key will display the previous five. Each menu is actually a record in the MENU database, and the menu number displayed is the record number for the menu. Accordingly, the number of a given menu may not be changed. If a new menu is desired, entering a RETURN will begin creation of a new menu. To delete a menu, the menu should be selected for editing and the DELete RECord function key pressed. Entering the number of an existing menu will move execution to the next screen, with data filled into the blanks for the menu you selected.

EDITING EXISTING MENUS

This is the menu configuration screen:

```
DATAFLEX MENU SYSTEM                                    MENU CONFIGURATION
=========================================+==========================

MENU NUMBER: __.          HEADER1: ________________________
                          HEADER2: ________________________

                          DEFAULT MENU: __. (on return)

   PROMPT                    ACTION                    PASSWORD
 1. _____________________  _____________________  ______
 2. _____________________  _____________________  ______
 3. _____________________  _____________________  ______
 4. _____________________  _____________________  ______
 5. _____________________  _____________________  ______
 6. _____________________  _____________________  ______
 7. _____________________  _____________________  ______
 8. _____________________  _____________________  ______
 9. _____________________  _____________________  ______

(N)ew Menu  (H)eader  (Q)uestions  (P)rint   (S)ave      {HELP IS
(I)nsert    (D)elete  (C)hange     (A)ppend  e(X)it  _    AVAILABLE}

_______________________________________________________  _ _
```

The first column of windows displays whatever prompts are in the record for existing menu choices. The column displayed under the word "PROMPTS" are the actual on-screen prompts displayed to the operator describing what the menu choices are.

The second column displays the commands which are passed to the operating system upon selection of the choice. As "ACTIONS" in the second column above, the menu system will execute any of the system commands described in the section of this manual on Manipulating Files except RUNPROGRAM. It will also execute another menu by entry of the word "MENU" followed by the number of the desired menu. Sub-menus are executed by use of this command. Entry of the CHAIN command followed by a rootname in this window will cause the menu system to execute any existing DataFlex configuration having that name.

The third column displays a password (if any) to protect the menu choices from unauthorized access. Since passwords are displayed by MENUDEF, MENUDEF must itself be password protected if passwords are used. Whoever has access to MENUDEF has access to all the passwords in the system.

The cursor initializes in the single-character blank at the lower right corner of the screen. If "N" is entered, the screen will clear and then redisplay the previous screen (list of menus) for selection of an unused number under which to create a new menu.

The "I" choice will query as to the choice number before which you would like to insert a menu choice.

The "H" choice will place the cursor into the blanks at the top of the screen, where the menu header may be modified. The header includes definition of the "Default Menu", which is the number of the menu which should be executed in the event that a null response (RETURN) is given to the choice query. Typically, the default menu will be the menu from which the menu being defined was called, but it need not necessarily be, especially for the master menu, which typically defaults to itself.

The "D" choice will ask which menu choice number you would like to delete, and do so upon entry of the number.

The "Q" choice will exit the screen temporarily and display the questions for the menu on another screen, which will be shown and discussed below. Questions are operator prompts which elicit further input from the operator relevant to the choice which was made, such as "Press RETURN when the printer is turned on and loaded".

The "C" choice will query which line of the menu you would like to change, and place the cursor in the blanks for that choice so that changes may be made.

The "P" choice will permit printing of the menu definition, which is very handy for occasional off-line reference to the structure of a menu.

The "A" choice places the cursor on the next blank choice line for the entry of an additional choice at the bottom of the menu.

The "S" choice causes an exit of the screen and writing of the edited menu to disk (making the entered changes permanent).

The "X" choice causes the question to appear: "BEFORE YOU EXIT, SAVE YOUR CHANGES? Y/N" to appear. Once this question is answered, the Menu Definition system will be left, and the menu from which it was called will appear.

OPERATOR QUESTIONS

Questions can be asked of the operator, and the responses included in the action of the menu, by the inclusion of the characters $#, where the first character is explicitly the dollar sign, and the second, represented by the # sign, is a digit 1 through 6, identifying the question number matching the question defined for the menu on the question screen shown below:

```
DATAFLEX MENU SYSTEM                                          QUESTIONS
=======================================================================

EXPLANATION: _________________________________________________________
QUESTION 1 : _________________________________________________________

EXPLANATION: _________________________________________________________
QUESTION 2 : _________________________________________________________

EXPLANATION: _________________________________________________________
QUESTION 3 : _________________________________________________________

EXPLANATION: _________________________________________________________
QUESTION 4 : _________________________________________________________

EXPLANATION: _________________________________________________________
QUESTION 5 : _________________________________________________________

EXPLANATION: _________________________________________________________
QUESTION 6 : _________________________________________________________

Enter the screen prompt for each question referenced in the ACTION
section of the other screen, or ESCAPE to return.

______________________ _
```

Each menu will support only six questions, but each defined question may be asked as many times as desired throughout the menu. Each defined question will display up to two lines to the operator, if needed, for an introductory explanation and the following actual question. The cursor initializes in the single-character blank in the lower right of the screen. Entry of a number for which a question already exists will place the cursor in the blanks for that question for editing of the data in the blanks. Entry of a number for which there is no question will place the cursor in the blanks for that number for entry of new data. Data for a question may be deleted with the field delete key, but there is no reason to delete such data until the number is needed for another question. Actual deletion of a question consists of removing references to its number from the action windows of the Menu Configuration Menu.

Each menu supports only nine choices, so that selection may be made by the entry of a single character, and to keep each screen display easy for the operator to scan. Extensive menu systems may nonetheless be constructed by thoughtful arrangement of sub-menus of related groups of functions called from a main menu.

ELEMENTS OF CONFIGURATIONS

There are several types of arguments (see Definitions) which can be acted upon by a DataFlex command. These serve the purpose of enabling the manipulation of various types of information both literally and symbolically. Arguments are established by DataFlex commands, file structure definitions or various forms of delimiters, and are identified by symbols.

The DataFlex commands which establish arguments are declarative in nature, i.e. they provide no action other than establishing the variable. These commands are treated separately in the chapter on "Definition Commands". There is a Definition command for each of the argument types, which are listed below:

ARGUMENT TYPES
- STRING
- NUMERIC
- DATE
- INTEGER
- INDICATOR

Most arguments have both a type and one of the classes listed below:

ARGUMENT CLASSES
- CONSTANT
- VARIABLE
- DATA BASE ELEMENT
- WINDOW

CONSTANTS are explicit data values that do not change. For example, 3 is a numeric constant; 12/12/83 is a date constant.

VARIABLES represent data values that can change in the course of a configuration's execution. It is maintained in the computer's memory, and is of a fixed length. A variable is "typed" as listed above, and can contain only information consistent with its type. If data is not of the proper type for a variable, DataFlex will convert it to the proper type when it is MOVEd. (see MOVE command) Variables in DataFlex are global in a configuration.

DATABASE ELEMENTS are the smallest discrete addressable components of the database. In DataFlex they are addressed with the syntax "FILENAME.FIELDNAME" where FILENAME is the DF_FILENAME established for the data file in the DataFlex FILEDEF Utility, and FIELDNAME is the TAG_NAME established for the data field in the DataFlex FILEDEF utility. Any database element existing on any addressable disk drive can be addressed from any configuration.

EXAMPLES:

```
          DF_FILENAME.TAG_NAME
          INVENTORY.QTY_IN_STOCK
```

WINDOWS are typed and formatted positions in DataFlex screen images. They are addressed by reference to the name of the PAGE of which they are a component, and their sequential position in that page. The sequence is established by numbering the windows from left to right, top to bottom in the PAGE; i.e. the top leftmost window is number 1. The number is combined with the name of the PAGE in the following format: PAGENAME.# where "#" is the sequential number of the window in the page. Symbolic names can be created for windows with the NAME command.

STRING CONSTANTS

A text string constant is a literal representation of a group of printable characters that is handled "as is" by the command processor. The string is defined to the DataFlex command processor by enclosing the group of characters in single ('string') or double ("string") quotation marks as delimiters. The form of the quotation mark selected to start the delineation of the text string must also be used as the marker for the close of the string. This is done to allow the use of either of the text string delimiters within a string.

EXAMPLES:

```
     'text string'   "computer"  'Press <ESC> to exit... '
     "Tom's file"    'In case of fire, yell "HELP"!'
```

Care must be taken to use the correct combination of beginning and ending quotation marks with strings that contain the alternate form. For example, the string 'I can't go!' would be interpreted by DataFlex as 'I can'. "I can't go!", however, would be interpreted correctly.

STRING VARIABLES

A string variable is a name which you designate to represent a string of characters which will have a consistent meaning (but with contents that vary) throughout your application (for example, "CUSTOMER_NAME"). The variable name must be declared (using the STRING command), and its length either specified in the command or not specified, in which case it defaults to 80 characters. The string variable must be declared before it is used. After declaration, the string variable can be referenced or accessed by name throughout a configuration. Variable names may be up to 80 characters in length, must begin with a letter, and may not contain a space.

If you don't specify your variable's length, but the contents are only four characters ("Bill"), DataFlex will still handle it as 80 characters long (with 76 spaces, in this case!). So for variables with short contents, you can save a great deal of memory space by defining your variables as short ones (see examples below). String variables can have more than 80 characters in them, too, if they're defined to. They may have up to 255 characters in them, if needed. But remember, variables defined as large will always be large, even when their contents are short. "Bill" would be 255 characters if it were the contents of a string variable defined as having length 255.

The command format for declaring a string variable is:

```
        STRING variable_name {string length}
```

EXAMPLES:

```
        STRING flag 1
        STRING response
        STRING an_input 10
        STRING my_entry 100
```

The strings "FLAG", "RESPONSE", "AN INPUT" and "MY ENTRY" are declared for a DataFlex configuration by the above commands. "FLAG" is defined as having a length of one (1) character, "RESPONSE" is defined to eighty (80) characters by default, since length isn't specified in its command, "AN INPUT" is defined to a length of ten (10) characters, and "MY_ENTRY" is defined to a length of one hundred (100) characters.

NUMERIC CONSTANTS

Numeric constants are single, specific numeric values that are used in expressions or other forms of numeric arguments.

EXAMPLES:

```
        1       234      100        100000     -50     10.52
        (diameter*3.14)  (FICA_WAGES*13.7)
```

On the first example line, the six numbers are simply that: numbers that can be used wherever and however useful or necessary. On the second example line, the values 3.14 and 13.7 are numeric constants used in expressions.

NUMERIC VARIABLES

A numeric variable is a named symbolic representation of a number that can be used as the subject of a DataFlex command's execution or expression evaluation. The numeric variable must be declared before it is used. After declaration, it can be referenced or accessed by name throughout a configuration. Only numbers may be stored in a numeric variable.

Multiple numeric variable arguments can be defined on a single definition line. The variable names must be separated by spaces.

FORMAT:

```
        NUMBER variable_name1 variable_name2 ... variable_namen
```

EXAMPLES:

```
        NUMBER quantity
        NUMBER amount_due credit
        NUMBER area_code phone_exchange phone_number
```

The allowable range of values for numeric variables in DataFlex is:
+ or - 99,999,999,999,999.9999

EXPRESSIONS

Arithmetic expressions are numeric type arguments that are evaluated as they are acted upon by DataFlex commands. Expressions do not have to be evaluated first and then placed into a variable. They are automatically evaluated by the DataFlex command processor when encountered during the execution of a configuration. The result of the expression is then processed by the command.

Any valid numeric representation may be a component of an expression including: numeric constants, numeric variables, numeric elements in the data base (addressed as filename.fieldname), date variables or data input from the keyboard by an operator and stored in a data window.

The following symbols are used to instruct the command processor how to perform the expression evaluation:

ADDITION	+
SUBTRACTION	-
MULTIPLICATION	*
DIVISION	/
RETURN GREATER THAN	>
RETURN LESS THAN	<
PARENTHESES	(...)

The first four operators above are self-explanatory. However, the last two expression operators require some explanation. Two numeric arguments are supplied to these operators, and depending upon the symbol selected, the greater or lesser of the arguments will be returned. For example: in 3>5, 5 will be returned; in 3<5, 3 will be returned. No modification is made to the data. These evaluations only make selections.

Expressions are formatted by enclosing the numeric arguments which make up the expression, with operators, in parentheses. The DataFlex command processor recognizes the parentheses and evaluates the expression, making the result available for command processing just as if a constant had been used in place of the expression.

EXAMPLES:

```
(arg1 + arg2)   (balance + sale)
(arg1 - arg2)   (balance - payment)
(arg1 * arg2)   (quantity * value)
(arg1 / arg2)   (principal / months)
```

Multiple levels of parentheses are supported to control the sequence of the arithmetic evaluation of the expression. In an expression that contains multiple parentheses, the portion of the expression contained in the innermost pair of parentheses will be evaluated first.

EXAMPLE:

```
(unit_cost)*(reorder_qty - qty_on_hand))
```

In this example, UNIT_COST would be multiplied by the difference between REORDER_QTY and QTY_ON_HAND because the difference is enclosed in parentheses. Without the parentheses, UNIT_COST would first be multiplied by REORDER_QTY, and QTY_ON_HAND subtracted from the product, a meaningless result.

Without parentheses in an expression, evaluation is left to right.

DATE VARIABLES

NOTE: DataFlex supports two date formats in different versions of the product. The two supported formats are MM/DD/YY and DD/MM/YY. We refer to the latter as the "Euro-date" format. For the purposes of this documentation we will refer to formatted dates as MM/DD/YY, even though Euro-date versions of DataFlex accept and return dates in the format of DD/MM/YY.

Dates are stored in DataFlex in Julian form as numbers. The displayed and printed date format MM/DD/YY which appears in screens and reports is a de-coded representation of the numeric storage format. A date variable can be defined to manipulate date information within a configuration by using the DATE definition command. Multiple date variables can be defined on a single argument definition line.

The data stored in the date variable is a Julian numeric value. Therefore, any operation requiring calculations on dates can be executed directly by reference to the variable without the necessity for type conversion. When dates are output to screens and reports, internal DataFlex routines which are flagged by the data type of the variable decode it automatically into the MM/DD/YY format.

FORMAT:

```
        DATE date_variable1 ... date_variable9
```

EXAMPLES:

```
        DATE due
        DATE today tommorrow yesterday invoice_due
```

DATE CONSTANTS

Date constants (expressed as MM/DD/YY) can be used as the subject of DataFlex commands, but they cannot be used within expressions. They can be placed in date variables using the DataFlex "MOVE" command for use in date variables.

EXAMPLES:

```
        LEGAL   --  DATE BIRTHDAY
                    MOVE 01/16/46 TO BIRTHDAY

        LEGAL   --  DATE BIRTHDAY
                    MOVE 01/16/46 TO BIRTHDAY
                    CALCULATE (BIRTHDAY + 365) TO NEXT_BIRTHDAY

        ILLEGAL --  CALCULATE (01/01/83 + 10) TO INVOICE.DUE

        LEGAL   --  DATE DATE_SOLD
                    MOVE 01/01/83 TO DATE_SOLD
                    CALCULATE (DATE_SOLD + 10) TO INVOICE.DUE
```

FILENAMES

The "df_filenames" of DataFlex data base files can be used as command arguments. Valid "df_filenames" that can be used as arguments must be established with the DataFlex FILEDEF utility (see the section of this manual on File Definition for a discussion of the df_filename conventions).

FORMAT:

```
        OPEN df_filename
```

"df_filename" in the format example is an argument for the OPEN command.

EXAMPLE:

```
        OPEN inventory
        CLOSE customer
```

The symbolic names of individual elements in a database can be used as arguments for a variety of DataFlex commands. A specific, formatted syntax is provided to represent these arguments, which are composed of the *filename* and *fieldname* of the data base element:

FORMAT:

```
        FILENAME.FIELDNAME
```

EXAMPLES:

```
        inventory.description
        customer.current_balance
        customer.credit_limit
```

INDICATORS

An indicator is established by issuing an INDICATOR command with the indicator name as the argument of the command. It takes on the characteristics of a "flag" which is tested by the configuration to determine whether a command line or group is to be executed. The INDICATOR command only establishes an indicator; it does not "set" it to a status.

FORMAT:

```
        INDICATOR indicator_arg1 ... indicator_arg9
```

EXAMPLES:

```
        INDICATOR pass fail new old change
```

See the chapter on INDICATORs for more information.

THIS PAGE INTENTIONALLY LEFT BLANK

DECLARING DATA ELEMENTS

There are five DataFlex Definition Commands:

STRING
NUMBER
DATE
INTEGER
INDICATOR

They serve to provide the configurator with a means of identifying typed variables to the DataFlex compiler. Once identified, the variable argument can be used at will throughout the configuration. The definition command which establishes a variable must precede the first use of the variable in a given configuration.

FORMAT:

```
        COMMAND variable_arg
```

EXAMPLE:

```
        STRING keyboard_input
```

The first three definition commands support the three data types which can be defined in a DataFlex file structure: ASCII strings, numbers (stored as packed BCD), and dates (Julian - stored as packed BCD numbers). In the case of the NUMERIC and DATE commands, up to nine (9) variables can be defined on a single command line.

STRING Command

If the maximum string length is not specified, a default maximum length of 80 characters is assigned by DataFlex. Use of defined string space is not dynamically allocated in memory; therefore, length specification can significantly conserve memory utilization. Strings consume their defined length plus 2 bytes in memory. The allowable range for string length specification is from 1 to 255 characters.

FORMAT:

```
        STRING variable_name {string length}
```

EXAMPLES:

```
        STRING flag 1
        STRING response
        STRING my_entry 100
```

The string variables FLAG, RESPONSE, and MY_ENTRY are declared for a DataFlex configuration by the above commands. FLAG is defined to one (1) character, RESPONSE is defined to eighty (80) characters by default, and MY_ENTRY is defined to a length of one hundred (100) characters.

If entry is subsequently made to a string variable which exceeds the declared length of the variable, excess input will be discarded, such as:

```
        STRING code 3
        MOVE "ABCDE" TO code
```

CODE will not contain ABCDE. It will contain only the first 3 characters, ABC. No error will be declared unless one is provided for the condition in the configuration.

NUMBER Command

Multiple numeric variable arguments can be defined on a single definition line. The variable names must be separated by spaces.

FORMAT:

```
        NUMBER variable_name1 variable_name2 ... variable_namen
```

EXAMPLES:

```
        NUMBER quantity
        NUMBER amount_due credit
        NUMBER area_code phone_exchange phone_number
```

A NUMBER variable consumes 10 bytes of memory, and has a value of zero automatically when it is declared. If there is sufficient memory for storing them, DataFlex can manage up to 32,767 active NUMBER variables at one time. The range of numbers in DataFlex is:
+ or - 99,999,999,999,999.9999

DATE Command

Multiple date variable arguments can be defined on a single command line.

FORMAT:

```
        DATE date_arg1 ... date_argn
```

EXAMPLES:

```
        DATE invoice_date
        DATE due_date first_reminder second_reminder
```

INTEGER Command

The INTEGER command allows the creation of a memory variable to contain an integer value. It differs from the NUMERIC variable in that the INTEGER variable consumes less memory for storage, and, of course, does not support fractional accuracy in calculations. However, the greatest value of integers in configurations is when they are used for counters, in FOR loops, and with the INCREMENT command. Some pre-defined integers provide access to DataFlex system variables (e.g. ERRLINE = the last error line).

The range available for integers is plus or minus 32,767. There is no INTEGER storage type in the data base; all numbers are stored on disk as binary coded decimals (BCD). In memory, however, a NUMBER data element consumes 10 bytes, while an INTEGER data element occupies only 2 bytes.

FORMAT:

```
        INTEGER argument1 ... argumentN
```

EXAMPLES:

```
        INTEGER my_age
        INTEGER qty_sold qty_returned qty_shipped
```

Integer variables are stored in an internal array in the order in which they are declared. When the DataFlex CHAIN command is executed, the integer array is <u>not</u> reinitialized. This provides the facility to pass integer values from one configuration to another without writing the values to a file.

The internal integer array is sequentially allocated as the variables are declared. Therefore, if it is desired to pass integer values in CHAINed configurations, we suggest that the same integer names be declared at the beginning of each configuration to provide for a consistency in variable referencing. When integer values are not passed between configurations, they should be initialized before use.

EXAMPLE:

```
        MOVE 0 TO sheep_counter  // initialize sheep_counter
```

INDICATOR Command

The INDICATOR definition command defines a special type of DataFlex argument which is used for conditional processing and program control. The INDICATOR command creates an "indicator".

The named indicator established as the argument of the INDICATOR command takes on the characteristics of a "flag" which is tested by the configuration to determine whether a command line or group is to be executed. Indicators are discussed fully in the section on "Conditional Execution".

FORMAT:

```
        INDICATOR indicator_arg
```

MOVE Command

MOVE provides the facility to "transport" data throughout a DataFlex application without regard for the type or location of the source or destination arguments. Data can be MOVEd from window to window, window to data base "element", element to window, element to element, expression to window or element, etc., etc. Where expressions are part of the MOVE command, they will be automatically evaluated with the result being MOVEd to the destination argument. Further, any required type conversions will also be executed automatically.

FORMAT:

```
        MOVE source_var TO destination_var
```

EXAMPLES:

```
        MOVE "DataFlex" TO string_arg
        MOVE 06/01/83 TO date_arg
        MOVE (invt.qty - qty_ordered) TO invt.qty
        MOVE screen.date TO invoice_date        // then...
        MOVE (invoice_date + 30) TO invoice.due
```

The MOVE command decodes the type of the destination variable and internally generates one of the following types of move commands for the command processor:

```
        MOVENUM for numbers
        MOVEINT for integers
        MOVESTR for strings
```

Neither variable may be a label or indicator. While the source may be a literal string or number, the destination variable may not be. The MOVE command will not update accumulators in the SUBTOTAL section of the REPORT macro. See the PRINT command for this purpose.

INCREMENT Command

The INCREMENT command operates on an integer variable, and is used as a counter for loops and repetitive operations. Literals, labels, and indicators may not be used in this command. It increments only by the integer 1.

FORMAT:

```
        INCREMENT integer_var
```

This command is equivalent to:

```
        MOVE (integer_var + 1) TO integer_var
```

...but is faster in operation.

CALCULATE Command

The CALCULATE command is used to evaluate an expression and move the result to a destination variable of the numeric type. When using the CALCULATE command you must determine if the destination argument is of the numeric type. If you are not sure, use the MOVE command instead. MOVE can also evaluate expressions.

An abbreviation of the CALCULATE command, "CALC" can be used interchangeably with the full command word.

FORMAT:

```
        CALC{ULATE} expression_arg TO destination_var
```

EXAMPLES:

```
        CALCULATE (12+6) TO g_total
        CALC (date_ord+cust.terms) TO invoice.date_due
        CALC (3.14*66.4) TO area
```

The destination variable may not be a label, indicator, constant, or ASCII variable. It may be an integer, numeric, or date variable. The CALCULATE command will not update accumulators in the SUBTOTAL section of the REPORT macro. See the PRINT command for this purpose.

DATA ENTRY

Application requirements for data entry, file maintenance and file inquiry, are efficiently met by the DataFlex ENTER command macro. ENTER provides a complete program skeleton for data entry using IMAGE formats and Flex-Keys. The ENTER macro can handle multiple related files, file update procedures, transaction creation and online editing. There are user-defined procedures for specific operations so the user can set up whatever auxiliary update procedures and error trapping may be required.

Since command macros are stand-alone program skeletons, they can only be used once in a configuration. Specific command statements are used within and around the command macros to provide functions to accommodate a variety of requirements.

The ENTER command invokes the ENTER macro, which is in fact an entire group of processing routines that perform the following functions:

1. ENTER data via CRT screen images, with validation, type checking, range checking and error trapping.

2. CREATE records in the data base with data entered through the screen image, and in the process, update related data base records, key indexes, etc., all without having to create routines and procedures for such activities.

3. FIND data in the data base by key field and display it on the formatted CRT "image". Single records or multiple related records can be "found" with a single key stroke. Data can then be inspected, scanned, edited, deleted.

4. EDIT existing records in the data base and in the process, properly update related data base records, key indexes, etc. without custom routines and procedures.

5. DELETE records from the data base while processing all required updates on-line.

6. CLEAR displayed data from the screen image.

The operation of the ENTER command macro assumes that the database files dealt with have previously been defined with the DataFlex FILEDEF or AUTODEF utilities.

Multiple records from different database files can be displayed with ease and flexibility by the ENTER macro. However, the structure of the ENTER command does impose the limitation that only one record from a given data file may be displayed on the screen at one time. It will not, for example, support a multi-item format like an invoice. Configurations requiring multi-line formats should be "programmed" without the use of the ENTER macro. The ENTERGROUP command is useful in applications requiring multi-line formatting. See the INVOICE.FRM demo file for an example of a programmed multi-line (non-ENTER) configuration.

A specific command structure is used with the ENTER macro, and there are a series of optional commands designed to control the formatting and actions which occur within the data windows of the screen image. Configuration examples at the end of this chapter can serve as a useful guide during the discussion of these commands.

ENTER CONFIGURATION STRUCTURE:

```
// ======================== IMAGE SECTION ===========================

/pagename1                      // start image section

    ---- first page of image ---

/pagename2                      // can have multiple images

    ---- following pages of image ---

/*                              // end of image section

// =================== PRE- ENTER COMMAND SECTION ===================

        OPEN file1              // open all files and
        OPEN file2              // initialize any variables used and
                                // give window formatting commands

// ============= INVOKE ENTER MACRO USING ENTER COMMAND =============

        ENTER file1 file2       // start ENTER mini program
                                // declare related files in order:
                                // "senior" to "junior"

// ======================== ENTRY SECTION ===========================

        AUTOPAGE page           // AUTOPAGE required only for
                                // second and following screens
        ENTRY field {window} {options}
        ENTRY ...    ....     ..... //one ENTRY for each window
        ENTRY field {window} {options}
        RETURN                  // required at end of ENTRY section

// =============== OPTIONAL USER-DEFINED PROCEDURES =================

        ENTER.SAVE:             // save procedure
        --- commands --         // save procedure commands
        RETURN                  // end of save procedure
        ENTER.EXIT:             // exit procedure
        --- commands ---        // exit procedure commands
        RETURN                  // end of exit procedure

ENTEREND                        // End of ENTER mini program

// ================ OPTIONAL POST-ENTRY PROCESSING =================
```

The above sections: Image Section, Pre-Enter Commands, Enter Command Line, Entry Section, User-Defined Procedures and Post-Entry Processing are discussed below. The section markers shown above are for instructive purposes only. They are not required in actual configurations.

ENTER IMAGE SECTION:

The **IMAGE SECTION** in a configuration using ENTER should start with one or more "fill in the blanks" forms for the operator. These images will be a window into the database for the file(s) used in the configuration. The same images can be used for all file maintenance functions and will use the Flex-Keys. The image form should contain prompting to inform the operator on the contents of the windows that are displayed. Each page of the image must be preceded with a pagename ("/" followed by the name of the page). Help screens may be included in the image section (see the HELP command). The image section of the configuration is terminated with a "/*".

ENTER PRE-ENTER SECTION:

In the PRE-ENTER section, all of the files that will be used by the ENTER macro must be OPENed. The OPEN command is further described in the section on "Manipulating Records". Any other DataFlex commands required to ready the system for a data entry session should be included here.

ENTER COMMAND LINE:

The **ENTER COMMAND LINE** must specify all of the database files that the ENTER macro will access during execution. The files listed on this line will be **automatically** SAVEd and CLEARed (when the operator presses the SAVE Flex-Key or presses <RETURN> in the last window of an image). Other files that are open may still be used within the ENTER macro but if it is required to SAVE and CLEAR any such file, those actions must be written into the configuration explicitly. FIND, RELATE and DISPLAY operations on records from such non-ENTER files will still be performed automatically.

The SEQUENCE in which the files on the ENTER command line are declared is significant. The "main" file, the one on which the primary file maintenance is being done, MUST be the first file in the list in a left to right sequence. Any file which is related TO by the main file must be declared to the right of the main file, and any file TO which that file relates should be listed next. This will put the files in order with relationships pointing to the right and is required to ensure that when a record is SAVEd it is properly attached to the file(s) to which it relates. This also ensures that DELETIONS will properly delete the relationships to the record(s) which are DELETEd.

The chapter on "File Definition" explains relationships among database files. Following the example given in that chapter, an ENTER command line for the Sales Transactions file would read "ENTER TRANSACTION PARTS CUSTOMER", and the PARTS and CUSTOMER database files would be updated through entries made into the TRANSACTION main database file (including new PARTs, new CUSTOMERs, and changes in both).

ENTER ENTRY SECTION:

The **ENTRY SECTION** is the primary interface with the data entry operator. The object of the entry section is to logically connect the data base elements with the data windows in the image. This is done with the ENTRY command. The ENTRY command handles the display of data, finding records, moving data into the record buffer and clearing the windows. When each of the above operations is executed, the entire entry section is scanned for ENTRY commands. One or more options can be placed on each ENTRY command line to provide formatting for the data controlled by the command.

The ENTRY command differs from any other DataFlex command in that it has several different functions and that they can be executed in any sequence. EXAMPLE: If the FIND Flex-Key is used in a window controlled by an ENTRY command, all of the ENTRY commands are scanned. If the FIND is successful, EVERY data element in that file for which there is an ENTRY command will display its contents in its data window.

Other commands (INDICATORs, GOTOs, etc.) can be included in the ENTRY section to control the sequence of operation or set conditional entries. SAVE commands should be avoided in the ENTRY section. When data is put into a data window with the ENTRY command, the data is JUST in the data window; it isn't moved into the record buffer until a SAVE is done. The CLEAR Flex-Key will clear windows controlled by ENTRY commands, but not those controlled by DISPLAY commands (see section on Direct Control of Data Entry for DISPLAY command).

The format of the ENTRY command is as follows:

> **ENTRY filename.fieldname window {options}**
> where "filename.fieldname" is any database element in an open file or an expression (for calculated fields).

In this instance, the brackets {} are to be actually typed in the configuration, contrary to the notation elsewhere in this manual where brackets are used only to signify optional command elements to the reader of the manual.
Format options for the ENTRY command are as follows:

FUNCTION	SYNTAX
DECIMAL POINTS	{POINTS=n} where "n" is the number of places to the right of the decimal point in the window. This can reformat a window or can be used to configure an integer-only numeric field by setting a value of zero (0) for "n".

AUTOFIND*	{AUTOFIND} Executes a FIND EQuals on the main index of which the data element on the ENTRY command line is a part. If the desired FIND is on a multi-segment index, the AUTOFIND should be on the last entry command for the segments that make up the index, so that all data necessary for FINDing has been entered prior to execution of the AUTOFIND. Record number may not be part of the index of which the AUTOFINDing data element is a part. Windows for data from related files will also be filled, provided there are ENTRY commands for those windows.
SKIP IF FOUND*	{SKIPFOUND} Display data in the window if a record is FOUND with key data matching the data in the window, and move the cursor to the next window. If no such record has been FOUND, leave cursor in window and accept operator input.
FIND REQUIRED*	{FINDREQ} Requires that an active record be in the buffer before the operator can pass this window in the image. The cursor will remain in this window until a successful FIND is completed. The BACKFIELD Flex-Key can move the cursor in reverse to execute the FIND on another window. Pressing the CLEAR Flex-Key will return the cursor to the first window on the image.
UPPER CASE	{CAPSLOCK} Converts all lower case alpha characters entered in the window to upper case.
REQUIRE ENTRY	{REQUIRED} The cursor cannnot be moved to the next data entry window until data has been entered into the current window.
RANGE CHECK	{RANGE = #1, #2} {RANGE = mm/dd/yy, mm/dd/yy} Provides an automatic check of entered data and declares an error and returns the cursor to the window if the date or numeric value entered does not fall within the specifications in the command. The first value is the low end of the range, and the second is the high end. The upper and lower values in the command are included within the valid range. Literal numbers and dates can be used. Variables may not be used.
VALIDITY CHECK	{CHECK="entry1¦entry2¦entryn"}{CHECK=string_var} Provides an automatic check of entered data and declares an error and returns the cursor to the window if the entry does not match any substring in the CHECK command. Where an eligible entry has fewer characters than the window, the excess characters must be specified as spaces (shown below as ƀ). The vertical bars (¦) shown below are recommended optional delimiters. Delimiters, if used, should be characters unlikely to be input erroneously by operators. The command may alternatively reference a previously-defined in-memory string variable (not a window or database element): {CHECK="YES¦NOƀ¦Yƀƀ¦Nƀƀ"}

FORCE PUT	{FORCEPUT} Puts data in the windows to the record buffer regardless of whether any data has been changed from that originally displayed. Normally, a put will not occur unless some data has been changed.
DO NOT PUT DATA	{NOPUT} Do not put data from the window into the record buffer. This protects data from being changed by the operator.
NO DATA ENTRY	{NOENTER} Data from the found record is displayed, but the cursor skips the window, preventing entry. NOENTER skips the window even if no record is found, unlike SKIPFOUND (see above).
DISPLAY ONLY	{DISPLAYONLY} Combination of NOPUT and NOENTER above. Data is not moved from the window into the record buffer, and the cursor skips the window.
RETAIN DISPLAY	{RETAIN} Maintain the data in window even if a CLEAR command is executed, but clear the window if two successive CLEARs are keyed.
RETAIN ALWAYS	{RETAINALL} Maintain data in window at all times; never clear.

* The commands with asterisks are all FINDing commands, and may be used only on windows relating to indexed database elements (see the chapter on "Designing Databases" for more information on indexing).

The command options, and the formatting options above can all be used cumulatively on an ENTRY command line:

```
     ENTRY file.field    page.5 {AUTOFIND,SKIPFOUND,CAPSLOCK}
```

The maximum length of any DataFlex command line is 255 characters. Multiple options on an ENTRY command line are all enclosed within the brace characters {braces}, separated by commas (,).

A special form of the ENTRY command is used where expressions are to be displayed. The default mode of the window becomes display only when this format is used.

FORMAT:

```
        ENTRY (expression) window_arg
```

EXAMPLE:

```
        ENTRY (date_today+30) screen.5
```

ENTRY AND AUTOPAGE

The ENTRY command can be used without specific reference to image windows by using the AUTOPAGE command. (AUTOPAGE functions automatically in the ENTER macro for the first screen image, and needs to be explicitly written only if there are two or more screens.) AUTOPAGE sequentially (left to right, starting witht the top line and working down) accesses the windows of the screen image named as the argument of the command (see the section of this manual on Direct Control of Data Entry), and permits the window references to be left off.

NOTE: The ENTRY command can ONLY by used within the ENTRY section of the ENTER macro or within the ENTERGROUP command.

Since ENTER is internally composed of a set of subroutines, a RETURN command is required to terminate the ENTRY section of the configuration.

If you don't need any of the user-defined procedures or post-entry processing, you may simply end the ENTER configuration with an ENTEREND.

The next page illustrates a totally straightforward, unimaginative, yet efficient ENTER configuration. It makes use of the AUTOPAGE function, so page window references are not necessary. It has all of the components of an ENTER configuration:

1. A labeled image

2. An OPEN command with filename argument

3. An ENTER command with filename argument

4. ENTRY commands with data element arguments

5. ENTRY commands terminated by a RETURN

6. An ENTEREND command to end the configuration

EXAMPLE FORMAT FOR ENTER

```
/LIST
-------------------------------------------------------------

                My Mailin' List

        Name: __________________________
     Address: __________________________
        City: _________________
       State: __          ZIP: _________

-------------------------------------------------------------
/*
// -- pre-enter --

        OPEN MAIL        // open data file

        ENTER MAIL       // start enter macro

// -- begining of entry section --

        AUTOPAGE LIST    // sequentially assign windows
                         // not actually required in this
                         // single-screen configuration
        ENTRY MAIL.NAME {AUTOFIND}
        ENTRY MAIL.ADDRESS
        ENTRY MAIL.CITY
        ENTRY MAIL.STATE {CAPSLOCK}
        ENTRY MAIL.ZIP

        RETURN           // end of entry section
                         // not actually required since next
                         // command is enterend

        ENTEREND         // end of enter macro
```

THE USER-DEFINED PROCEDURES

Many ENTER configurations will need to perform updates in related files, have error checking or delete protection built in, etc. These operations are accommodated by five user-defined procedures which are automatically executed when the corresponding DataFlex Flex-Key is pressed during the execution of the ENTER configuration. The User-Defined procedures are:

ENTER.SAVE:	executed on a SAVE command
ENTER.DELETE:	executed on a DELETE command
ENTER.EDIT:	executed on an EDIT and on a DELETE command
ENTER.CLEAR:	executed on a CLEAR command
ENTER.EXIT:	executed on an EXIT command

These User-Defined procedures are automatically called by the ENTER command macro, but they are functionally no different from any other labeled DataFlex sub-routine called by your own GOSUB commands. The inclusion of the colon ":" at the end of the procedure name is required to identify them as labels; omitting the colon will cause an error.

If an ERROR is declared within any of the procedures the associated function will be disabled.

The ENTER.SAVE procedure is executed whenever a new record is created or an existing record is edited.

The ENTER.DELETE procedure is executed whenever an existing record is deleted.

The ENTER.EDIT procedure is executed whenever an existing record is deleted OR an existing record is edited. This works well with the save procedure to keep the consistency of counters and totals in a related file.

The ENTER.CLEAR procedure is executed whenever the CLEAR Flex-Key is pressed.

The ENTER.EXIT procedure is executed whenever the escape key is hit and is primarily intended to allow conditional disabling of the escape key by declaration of an error.

Each of the User-Defined procedures must be individually terminated with the use of a RETURN command (see the example on the next page).

The configuration below illustrates the use of the ENTER.SAVE and ENTER.EDIT procedures, which are executed every time a record is saved and edited/deleted, respectively. An application requirement for a mailing list uses only one file for the list information. However, it is desired to maintain an on-line counter of how many records are in the file. To do this we will establish a field in the system file (SYSFILE.COUNTER) that contains the number of records in this file. To maintain the count correctly, the counter must be incremented when a record is created; when a record is deleted, the counter must be decremented, and when an edit is done, the counter must be left unchanged.

```
/LIST
------------------------------------------------------------

               My Mailin' List

        Name: __________________________
     Address: __________________________
        City: _________________
          ST: __        ZIP: _________

------------------------------------------------------------
/*
// only those parts of this example that are changed have comments

        OPEN MAIL
        OPEN SYSFILE              // open other file.

        ENTER MAIL
           AUTOPAGE LIST
                ENTRY MAIL.NAME    {AUTOFIND}
                ENTRY MAIL.ADDRESS
                ENTRY MAIL.CITY
                ENTRY MAIL.ST      {CAPSLOCK}
                ENTRY MAIL.ZIP
        RETURN

// The ENTER.SAVE procedure will execute whenever a record is saved.
// When a record is created, only this procedure is executed,
// incrementing the counter.
ENTER.SAVE:  CALC (SYSFILE.COUNTER+1) TO SYSFILE.COUNTER
        SAVE SYSFILE
RETURN          // return from save procedure

// The ENTER.EDIT procedure is executed whenever a record is deleted
// or edited.  For a deletion, it decrements the counter.  It also
// decrements the counter in an edit, but true edits are followed by
// SAVEs, which execute ENTER.SAVE, offsetting the decrement to zero.

ENTER.EDIT:  CALC (SYSFILE.COUNTER-1) TO SYSFILE.COUNTER
        SAVE SYSFILE
RETURN          // return from delete procedure

        ENTEREND
```

ENTDISPLAY Command

The ENTDISPLAY command causes all ENTRY data elements for the data file named in the argument to display on the screen, along with all those in other related files. Its apparent effect is exactly as though the SUPERFIND Flex-Key had been pressed by the operator. It may be used in the ENTRY section of an ENTER or ENTERGROUP configuration and/or in a KEYPROC (for the FIND key, for example).

FORMAT:

```
        ENTDISPLAY <df_filename>
        Where <df_filename> is the name of the data file from in which
        the displayed record is to be found.
```

EXAMPLE:

```
        FIND GE mail BY INDEX.1
        ENTDISPLAY mail
```

ENTDISPLAY would be used, for example, in an alphabetic FIND such as the one in the configuration above which is done with the AUTOFIND option on the ENTRY MAIL.NAME line. If it is desired, however, to do a FIND greater than or equal to (frequently desirable when the FIND target's spelling is not exactly known), the AUTOFIND option will not allow this, since it only does exact FIND equals. For that, the following configuration would be appropriate (note the differences):

```
/LIST
-------------------------------------------------------------

               My Mailin' List

        Name: __________________________
     Address: __________________________
        City: _________________
          ST: __        ZIP: ________

-------------------------------------------------------------
/*
// only those parts of this example that are changed have comments
// this configuration will always find a record

OPEN MAIL
ENTER MAIL
   AUTOPAGE LIST
   ENTRY MAIL.NAME       // AUTOFIND removed here
   FIND GE MAIL.NAME     // FIND greater than or equal to added here
   ENTDISPLAY MAIL       // ENTDISPLAY command
   ENTRY MAIL.ADDRESS
   ENTRY MAIL.CITY
   ENTRY MAIL.ST         {CAPSLOCK}
   ENTRY MAIL.ZIP
RETURN

ENTEREND
```

Another use of ENTDISPLAY is, in a multiuser environment, after a REREAD command, to display the REREAD record for inspection.

POST-ENTRY PROCESSING:

After the end of the User-Defined procedures, the command ENTEREND must be used to terminate the ENTER macro. When the escape key is pressed, the ENTEREND command will close out the ENTER macro. At this point you may ABORT configuration, CHAIN to another configuration, or proceed with any other sequence of DataFlex commands.

ENTER -- SEQUENCE OF OPERATION:

An understanding of the order in which the ENTER macro executes specific functions will aid in the development of more complex configurations. Steps that involve specific multiuser operations are identified. DataFlex systems in which single-user mode was selected (see SETSCREEn procedures) will not perform these multi-user functions.

CREATING A NEW RECORD:

a) Clear screen and clear record buffers.
b) Operator fills in the data windows.
c) Operator hits the SAVE key or exits the last image window.
d) Record is automatically locked and related records are re-read from the database (multi-user systems only).
e) Data in data windows is moved to record buffer(s) according to the ENTRY commands.
f) ENTER.SAVE procedure is executed (including any parts defined by the user).
g) Records are ATTACHed and SAVEd to disk.
h) All records are unlocked (multi-user systems only).
i) Screen and record buffers are cleared for new entry.

EDITING AN EXISTING RECORD:

a) Clear screen and clear record buffers.
b) Operator FINDs a record in the main file by any available key.
c) Related record(s), if any, are found and displayed.
d) Operator changes data in window(s).
e) Operator SAVEs record.
f) Records are locked and all active records are re-read from disk (multi-user systems only).
g) ENTER.EDIT procedure is executed (including any parts defined by the user).
h) Data is moved from CHANGED data windows to record buffer.
i) ENTER.SAVE procedure is executed (including any parts defined by the user).
j) Records are ATTACHed and SAVEd to the data base.
k) All records are unlocked (multi-user systems only).
l) Screen and record buffers are CLEARed for new entry.

DELETING A RECORD:

a) Clear screen and clear record buffers.
b) Operator FINDs a record in the main file by any key.
c) Related record(s), if any, are found and displayed.
d) Operator presses the DELete RECord key.
e) Records are locked and all RELATED records are re-read from the data base (multi-user systems only).
f) ENTER.DELETE procedure is executed (including any parts defined by the user).
g) ENTER.EDIT procedure is executed (including any parts defined by the user).
h) Record in main file is deleted.
i) Related records are saved.
j) All records are unlocked (multi-user systems only).
k) Screen and record buffers are cleared for new entry.

ENT$PERMISSIVE INDICATOR

The ENTER macro is designed to prevent errors of the sort which could arise from operator sequences such as the following:

a. Operator finds a record and one or more related records by use of SUPERFIND (the display of data from records in multiple files related to each other is not necessarily apparent to the operator);

b. Places cursor in a key field in a related-to record;

c. Presses FIND and FINDs another record in the related-to file, bringing it into the record buffer with the earlier-found relating record; and

d. SAVES this combination of possibly-unrelated records as though they were related to each other.

What actually happens in this instance is that the ENTER.EDIT procedure is executed on the NEW record instead of the OLD one. This makes the totals and counters in both the old and new related-to files incorrect.

This possibility is eliminated by the internal resetting of the FIND key to function as a SUPERFIND (i.e., it will find only combinations of records actually related to each other) after a SUPERFIND has been done, just as the NEXT RECord and PREVious RECord keys do. Similarly, after a SUPERFIND, the AUTOFIND option is switched off on all but the main file, so that those fields may be edited without interference from undesired AUTOFINDing. If the option of editing the AUTOFINDing fields is not desired, editing can be prevented by addition of the NOPUT option to such fields.

The ENT$PERMISSIVE indicator will allow operational sequences like the one described above, which is normally an error, by not resetting the functioning of the FIND key and the AUTOFIND option. For those situations in which it is a permissible sequence, the statement

"INDICATE ENT$PERMISSIVE TRUE" should be given prior to the beginning of the ENTER or ENTERGROUP macro. This option should not be used in transaction-oriented applications in which counters are provided and/or transaction totals are maintained.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORTING AND OUTPUT

Two areas of activity consume a great portion of the development time for application software: (a) data entry and file maintenance; and (b) file output for reports, labels, preprinted forms and files. The DataFlex ENTER command macro provides a predefined facility for dealing with the requirements in the first area efficiently; the REPORT command macro takes care of the second.

The REPORT command macro consists of an integrated set of selectable predefined output routines that can be chosen as needed. You can use the selected routines with DataFlex commands and arguments relating to the requirement at hand to "fill in" an output "image" and then send the image to the output device. The REPORT macro may be used only once in a configuration. The REPORT macro is a convenience, usable for most normal reporting needs. It is not a necessity. The chapter on "Direct Reporting and Output" includes a simple report that does not use the REPORT macro.

Likewise, the DataFlex utility QUERY **generates** REPORT configurations, and one of the best ways to learn how REPORT works (if, indeed, there is need for you to do so), is to "build" your report in QUERY and then study the source file generated by your work. Unless you need to report from multiple related files, it is likely QUERY can satisfy all your reporting needs, and even if it can't, QUERY is an excellent way to begin a REPORT configuration, by producing a source file which you may then edit to contain all you need.

A REPORT configuration is divided into two major areas: image lines and command lines. The image lines specify the headings, window prompting, window formatting, etc. The command lines contain commands that instruct the REPORT command macro how to "fill in" data in the image area(s) for output.

In this manual, we refer to a "report" as a generalized formatted output operation which may variously encompass formatted reports (in fact), formatting for pre-printed forms, labels, file output, etc.

Sections of REPORT

The image and command areas of a REPORT configuration contain corresponding labeled "sections" which address different functional requirements of a report. For example, one of the "sections" is the "HEADER". The HEADER section of the image defines how the top of each page of the report is to look; perhaps your company's name, the name of the report, its date, page numbering, reporting sequence of the data, and so on. Commands in the HEADER command section provide any needed processing for data which is positioned and formatted in the HEADER image section of the report. If there are data windows that are filled in from variables in the configuration, such as for run date or page number, the HEADER command lines "fill in" the necessary data in the windows. When data has been filled into a section, it is OUTPUT.

The labeled "sections" of a REPORT command macro are:

```
        /HEADER
        /SUBHEADER#
        /BODY
        /SUBTOTAL#
        /TOTAL
```

The diagonal character (/) is part of the label syntax. The "#" after SUBHEADER and SUBTOTAL indicates a digit, 1 through 9, by means of which levels (subtotal breakpoints) can be established. The use of the sections and the names for both the image and command sections is critical.

DO NOT attempt to use them in a different order or change the names.

Within the BODY section, note the PRINT commands. These commands have two functions: (a) they move data elements from the database into their respective positions in the section image(s) for subsequent OUTPUTting; and (b) they update any subtotalling or totalling registers which may be assigned to the windows to which the data elements are PRINTed. This command is the REPORT equivalent of MOVE or DISPLAY, and should always be used in REPORT instead of the other two commands, since MOVE and DISPLAY do not update subtotal and total registers.

The following simple report sends five data fields to the default output device (the printer) on a single line using only the HEADER and BODY REPORT sections.

```
/HEADER
                              CUSTOMER  REPORT

CUSTOMER                       DISC  PURCH/MO  PURCH/YR    PROFIT

/BODY RESIDENT
______________________________ __. ______.__ ______.__ ______.__
/*
OUTFILE
OPEN CUST 1              // opens customer file, specifies index 1
REPORT CUST BY INDEX.1 // invokes REPORT for "cust" file, INDEX 1
SECTION HEADER           // defines HEADER section
  OUTPUT HEADER          // sends HEADER section to printer
SECTION BODY             // defines BODY section
  PRINT CUST.CUSTOMER    // moves data to window
  PRINT CUST.DISCOUNT    // moves data to window
  PRINT CUST.PUR MONTH // moves data to window
  PRINT CUST.PUR YEAR    // moves data to window
  PRINT CUST.PROFIT      // moves data to window
  OUTPUT BODY            // sends BODY section to printer
REPORTEND                // ends REPORT macro
ABORT                    // ends configuration, return to op. system
```

Several necessary components of all REPORT configurations are shown here:

1. A screen image (this one has 2 pages), followed by the "/*" marker. The BODY page has the RESIDENT option elected.

2. The OUTFILE command, required to initialize the output device (the default device in this case) for the configuration.

3. The OPEN command, which opens the main data file for access by its index 1. The definition for that file (CUST) created in FILEDEF are now available to the configuration.

4. The REPORT command, which invokes the REPORT macro for the execution of the commands that follow. The filename referenced on the REPORT command line places the specified file under control of the macro. The index referenced is the index that will be used to determine the sequence of the output of the data.

5. SECTION HEADER and SECTION BODY label the appropriate command sections. Since there is no variable or data base information PRINTed into the HEADER section, only the OUTPUT command is required to execute the HEADER. PRINT commands are used to move data into the image in the BODY section, from which it is OUTPUT. Only the PRINT command will properly update subtotal accumulators (MOVE will not).

6. REPORTEND is the required terminator for the REPORT command; ABORT then terminates the configuration.

DataFlex "invisibly" provides an AUTOPAGE command at the beginning of each section, and a RETURN command at the end.

The BODY of the report is the main section for general purpose output. For example, in that only the necessary "sections" of the REPORT command macro are used in a configuration, a report comprised of one line of data with no totaling or subtotaling might use only the HEADING and BODY Sections. Any of the sections may have multiple lines, including the BODY section, which would mean that each record reported would be reported on several lines, often a very desirable feature (as in the case of mailing addresses). The maximum line length which can be reported is 255 characters, but in all likelihood, your practical limit will be the line length of your output device (printer or screen).

Subtotalling and Totalling

The DataFlex REPORT command macro supports up to nine levels of subtotaling. The SUBHEADER# and SUBTOTAL# are linked in the processing of subtotal breaks. For example, SUBHEADER3 is the subheading column headers and labels that would appear at the top of information subtotaled in the third level of subtotalling. Totalling and subtotalling are discussed further later in this chapter.

The TOTALs section accumulates running cumulative totals for the entire output and, if total printing is provided for in your configuration, prints them at the bottom of the report.

It is not necessary to use all of the REPORT macro "sections" in a configuration. Only those sections needed for the desired report need be used. Every image section used, however, must have a matching command section, even if all that command section does is OUTPUT the image itself (use the OUTPUT command).

SUBTOTAL BREAKPOINTS

Subtotals are defined by declaring subtotal BREAKPOINT fields on the REPORT command line. The declared fields must be among those which make up the index named on the command line (which will determine the sequence of the report). NOTE: **If the INDEX does not include the same fields as those on which the breakpoints are based, in the same order, subtotal groupings will be incomplete, and breaks will occur randomly in the report.**

FORMAT:

```
        REPORT filename BY INDEX.# BREAK fil.fld1 fil.fld2
```

"FIL.FLD#" is used in the format above to save space in signifying a filename.fieldname element. The "FIL.FLD#" declarations following the BREAK command set the breakpoints for the REPORT configuration. Each "fil.fld#" is assigned a number sequentially as it is declared. This number then becomes the breakpoint number corresponding to the SUBHEADER# and SUBTOTAL# command section labels in the configuration. In this example fil.fld1 becomes breakpoint 1. The rightmost (higher numbered) breakpoints represent the subtotals that change the most. Breakpoint 1 changes the least. The maximum number of breakpoints allowed in subtotalling in the REPORT macro is 9.

When a change occurs in a field defined as a breakpoint, the SUBHEADER# and SUBTOTAL# command groups corresponding to the number of the breakpoint where the change occurred, and all higher-numbered subheaders and subtotals will be executed. The subtotalling registers (and thus the total register) are updated only with PRINT commands in the BODY section of the report. The MOVE and CALC commands will not accomplish this function.

EXAMPLE:

Suppose we had a data file, "SALES", containing sales transactions on various dates with customers in different states. Field 1 is Date, Field 2 is Customer Name, and Field 3 is State (there would probably be other fields also). The correct command to yield a report of transactions by date with subtotals by Customer, and subtotals by State, would be:

```
        REPORT sales BY INDEX.2 BREAK sales.3 sales.2
```

For this to work, INDEX.2 would have to be made up of State plus Customer plus Date (3 segments). Date is in the index for output sequence only--it is not a breakpoint. For more information on indexing, review the section of this manual on "Designing Databases".

When REPORT selections and breakpoints are processed, only the record from the file named in the command has been loaded into the buffer. If selections or breakpoints are based on fields in files to which the main file relates, a "RELATE file_name" command should be included in the selection section (see below under "Selections") of the configuration to bring the records from the related files into the buffer.

As a side effect to placing a RELATE command in the selection section, the correct records in related files will no longer be available in the subtotal sections. Thus, no file elements should be accessed or saved in the SUBTOTAL command section when there is a RELATE in the selection section. To overcome this situation, data can be PRINTed to windows in the SUBTOTAL image section while in the SUBHEADER command section, where the correct records will be active.

The example format below displays all of the possible sections of a REPORT macro configuration. They can all be used, or only those needed for a given requirement can be used. It should be kept in mind that this sample format represents only a portion of a complete configuration, and that portions of the configuration before and after the REPORT macro may query the operator for input (such as the date, or which report is to be run), and that other commands after the REPORTEND may involve an ABORT, the selection of another report, or anything else normally provided by a configuration (except another REPORT macro within the same configuration).

```
/HEADER
                           report title image
/SUBHEADER1
                            subheader1 image
/SUBHEADER2
                            subheader2 image
/BODY
                               body image
/SUBTOTAL2
                             subtotal2 image
/SUBTOTAL1
                             subtotal1 image
/TOTAL
                               total image
/*
OUTFILE
OPEN filename1 #
OPEN filename2
MOVE # TO PAGEEND
REPORT filename1 BY INDEX.# BREAK filename1.field filename2.field
INDICATE SELECT AS file.field mode data element
SECTION HEADER
        commands
SECTION SUBHEADER1
        commands
SECTION SUBHEADER2
        commands
SECTION BODY
        commands
SECTION SUBTOTAL2
        commands
SECTION SUBTOTAL1
        commands
SECTION TOTAL
        commands
RETURN
RPT.KEYPRESS:  subroutine commands
RETURN
REPORTEND
```

The next page contains a REPORT configuration from a real application, and is shown as an example of REPORT functions in actual use. It demonstrates subtotals and 1 level of breakpoints.

```
/HEADER
                         INVENTORY MOVEMENT REPORT
=====================================================================
/SUBHEADER1
VEN: _0. ITEM: _______________ DES: ________________________________
     STORE ID: _______________   QTY: ___.  PRICE $______.__

QTY-SOLD   INVOICE          PRICE        AMOUNT        COST      PROFIT
/BODY RESIDENT
_______.   _______       ___,____.__  ___,____.__ ___,____.__ ___,____.__
/SUBTOTAL1
--------                              ---------   ---------   ---------
_______.   TOTALS THIS PART ID        ___,____.__ ___,____.__ ___,____.__
--------                              ---------   ---------   ---------
/TOTAL
                                AMOUNT          COST         PROFIT
   GRAND TOTALS:           _$___,___.__  _$,___,___.__  _$____,___.__
                           ============  =============  =============
/*
OPEN INVT
OPEN ITEMS 2
OUTFILE
REPORT ITEMS BY INDEX.2 BREAK ITEMS.PART_ID // This is REPORT line.
SECTION HEADER                              // No selections.  This
        OUTPUT HEADER                       // report is global.
SECTION SUBHEADER1
        PRINT INVT.VENDOR                   // illustrates the use of
        PRINT INVT.VENDOR PART_ID           // default formatting.
        PRINT INVT.DESCRIPTION
        PRINT INVT.PART_ID
        PRINT INVT.QUANTITY
        PRINT INVT.LIST_PRICE
        OUTPUT SUBHEADER1
SECTION BODY
        PRINT ITEMS.QUANTITY TO BODY.1      // illustration of the
        PRINT ITEMS.INVOICE TO BODY.2       // use of PRINT TO
        PRINT ITEMS.PRICE TO BODY.3
        PRINT ITEMS.AMOUNT TO BODY.4
        PRINT ITEMS.COST TO BODY.5
        PRINT (ITEMS.AMOUNT-ITEMS.COST) TO BODY.6
        OUTPUT BODY
SECTION SUBTOTAL1                           // The first line of this
        SUBTOTAL BODY.1                     // section in the image
        SUBTOTAL BODY.4                     // above is dashes to
        SUBTOTAL BODY.5                     // print at the bottom of
        SUBTOTAL BODY.6                     // the columns, not under-
        OUTPUT SUBTOTAL1                    // scoring for windows.
SECTION TOTAL
        SUBTOTAL SUBTOTAL1.2
        SUBTOTAL SUBTOTAL1.3
        SUBTOTAL SUBTOTAL1.4
        OUTPUT TOTAL
REPORTEND
ABORT
```

PRINTING TOTALS AND SUBTOTALS

Accumulators are maintained for every REPORT image NUMERIC window. A cumulative (running) total can be printed for any window at any time in a REPORT or, if a window is used for subtotaling, its value can be printed, after which it is automatically cleared for the next subtotal sequence.

The running total is constantly available as a variable by appending a per cent sign (%) to the end of any window_arg. For example, to put the accumulated value of BODY.3 in a window:

```
        PRINT BODY.3%   // prints running total to next window
```

For subtotaling, the command SUBTOTAL is used with window_arg:

```
        SUBTOTAL subheader.2
```

The action of SUBTOTAL is to print the total currently in the accumulator, and then MOVE 0 into the accumulator to initialize it. Subtotal accumulators may not be addressed by use of the WINDOWINDEX or FIELDINDEX commands.

Calculations and string manipulations on data in formatted windows should be avoided, since the formatting can change the apparent "value" of the contents for purposes of the calculation or manipulation. Calculations and manipulations should be done on the source data (database element, literal string, in-memory variable, etc.) from which the window was filled to begin with, with replacement of the results to the window, if desired for output purposes. In REPORT, however, if BODY.4, for example, happens to be a formatted window, the running total accumulator BODY.4% need not be considered formatted, as it is not updated through the window, but rather from the same source as the window.

SELECTIONS

Selection is a report specification calling for a report of only a subset of all the records in the database to be listed. For example, in accounts receivable, you may wish statements to be printed only for those accounts having balances over 30 days old.

The REPORT macro provides for selection of records to be printed through the use of a pre-defined indicator, SELECT. Selections are accomplished by INDICATE commands which set REPORT's predefined indicator, SELECT, TRUE or FALSE according to the report's selection criteria. The selection section is not mandatory if no selections are desired (all records in the database are to be reported). Indicators are more thoroughly explained in the manual section on "Conditional Execution."

The selection commands, if used, should be placed immediately after the REPORT command line. Any DataFlex command(s) can be used within the selection section, but the most common is one or more INDICATE statements to set SELECT. If values in the current record set SELECT

TRUE, the current record will be printed. If values set SELECT FALSE, the current record will be bypassed.

It is frequently desirable to prompt input from the operator at run time to establish some or all of the selection criteria. These questions should be asked before the REPORT command. The responses to these questions are then used in the selection section in the INDICATE statements.

EXAMPLE:

```
        /BODY
                ---- body image ----
        /*
        NUMBER VENDNUM
        INPUT "ENTER VENDOR NUMBER: " VENDNUM
        REPORT INVT BY INDEX.1
        INDICATE SELECT AS INVT.VENDOR EQ VENDNUM
        SECTION BODY
                ---- body commands ----
        REPORTEND
```

This would do a report for the single vendor number entered by the operator at the beginning of the report run.

You may use indexes to speed up a report by "jumping" into the report when the value of the selecting field satisfies the selection criteria and back out again when it no longer does. This can be done only if the selection field is the primary segment of the index by which you are reporting. To jump into the index at a specific point, MOVE the LOWEST value of the selection into the record buffer BEFORE the report. This will start the report at the next record in the sequence of the index by which you are reporting.

To jump out of a report when you are past any records that could be selected, place a "RETURN END.OF.REPORT" command in the selection section. This will abort the report and print the totals. If totals are not desired where the report is aborted, the proper command is END$OF$REPORT.

EXAMPLE:

In the Inventory Movement Report above, the vendor number is the first field segment of INDEX.1 so we can jump into and out of the report index. In the following example, the report would cover only the vendor entered in response to the prompt, and would take much less time to run than the selection routine illustrated above, since instead of actually checking each record for a match with the desired vendor number, the routine uses the index to FIND the right vendor record with that vendor number.

```
        /BODY
                ---- body image ----
        /*
        NUMBER VENDNUM
        INPUT "ENTER VENDOR NUMBER: " VENDNUM
        MOVE VENDNUM TO INVT.VENDOR         //jump in here
        REPORT INVT BY INDEX.1
        INDICATE SELECT AS INVT.VENDOR EQ VENDNUM
        [NOT SELECT] RETURN END.OF.REPORT   //jump out here
        SECTION BODY
                ---- body commands ----
        REPORTEND
```

Since the selection section is executed for every record, it should be kept as small and fast as possible. Any calculation or string manipulation which can be accomplished outside the selection section will improve speed. Expressions should be evaluated outside the selection section, if possible, and the results represented in the selection section as numeric variables. Numeric variables are faster than numeric windows, and string manipulation commands in the selection section have the greatest potential of all types of operations to slow down execution. Calculations and string manipulations can be done in the HEADER command section, any of the SUBHEADER command sections, or entirely outside (prior to) the report macro, as necessary.

It is possible to make compound selections in reports, with various AND and OR relationships among the criteria. Refer to the section of this manual on "Conditional Execution" for the full range of possibilities. The following AND example shows how all customers with balances over $1,000 AND amounts over 90 days due would be selected (NINETY_DAYS_AGO is a date variable calculated by subtracting ninety from the run date before the selection section):

```
        INDICATE SELECT AS INVOICE.DATE LT NINETY_DAYS_AGO
        [SELECT] INDICATE SELECT AS CUST.BAL GT 1000
```

OR combinations can be established by inverting the SELECT indicator with a NOT, as in the following example, which would report all customers with EITHER balances over $1,000 OR balances over 90 days old:

```
        INDICATE SELECT AS INVOICE.DATE LT NINETY_DAYS_AGO
        [NOT SELECT] INDICATE SELECT AS CUST.BAL GT 1000
```

The GROUP ANY/ALL AND/OR ANY/ALL facilities can also be utilized in setting the SELECT indicator for your particular requirements, as described in the section of this manual on indicators.

PAGE CONTROL

Page control is handled with three predefined variables:

PAGEFEED offers three way control of how the page eject will be determined: -1 = single sheet; 0 = hardware formfeed; or ## = linecounter where ## is the total number of lines per page (physical page height from perforation to perforation). The command with no argument is interpreted as though its argument were 0 (FormFeed).

PAGEEND contains the number of printable lines per page. The default is 55. A value of zero (0) in this variable will cause continuous output with no page breaks. This is useful, for example, for output to a data file from which it is not intended to print directly.

LINECOUNT contains the number of lines output as of any given moment. There is not normally occasion to manipulate or reset the value of this variable, since it is used by the two others listed above.

These variables are global to DataFlex. They may be used with any kind of outputting, not exclusively under the REPORT macro. Use MOVE commands to put the desired values into these variables for page control immediately before the REPORT command.

This example illustrates operator-entered selections and page control:

```
/PROMPT

        List NAMES from which STATE: __
/HEADER
                NAMES FROM STATE OF: __

.............NAME............      .....CITY......

/BODY
______________________________      _______________
/*

FORMAT PROMPT.1 {CAPSLOCK}
STRING TEMP 1
ACCEPT PROMPT.1               // get operator input
CLEARSCREEN
OUTFILE 'CON:'                // output to the screen
OPEN LIST 1
MOVE 22 TO PAGEEND            // page length is # of lines on screen
MOVE -1 TO PAGEFEED           // stop at the bottom of each screen

REPORT LIST BY INDEX.1        // REPORT command

INDICATE SELECT AS LIST.STATE EQ PROMPT.1

SECTION HEADER
        PRINT PROMPT.1        // print state selection to heading
        OUTPUT HEADER
SECTION BODY
        PRINT LIST.NAME       // print data
        PRINT LIST.CITY
        OUTPUT BODY

REPORTEND

INPUT "PRESS RETURN TO CONTINUE" TEMP
ABORT
```

OUTPUT DESTINATION

Unless directed elsewhere, the default output device for the REPORT macro is the system LST: device (printer). Output can be redirected to any valid sequential system device or file by placing an argument on the operating system command line when the configuration is executed (FLEX report destination). This form of command can, of course, be included in menus for operator selection, with or without querying for filename or device input. Alternatively, the output device or file can be "hard coded" in the configuration on the OUTFILE command line immediately above the REPORT command line. Used in this way, literal destinations ("CON:", "MERGFILE", etc) must be embedded in quotation marks. Arguments without quotes will be considered string variables, which can be used to pass operator input in response to query prompts at the beginning of the configuration. Legal devices on most systems include CON: (CRT screen), PUN: (often used for modems), LST: (the printer), and any legal filename.

OPERATOR ABORT

The REPORT macro's default mode of operation causes a report to abort if any key is hit by the operator while the report is executing. This mode may be disabled entirely or modified to ask the operator a confirming question when a key is thus hit. To do this, provide a substitute for the predefined subroutine called "RPT.KEYPRESS", optimally just before the REPORTEND command. The illustrative subroutine below would give the prompt "PRESS <ESC> TO ABORT, ANY OTHER KEY TO CONTINUE", and then, depending on which key was pressed, either resume execution at the predefined procedure "RPT.LOOP" or abort via the predefined procedure "END.OF.REPORT". (Pressing any key other than ESCape actually would cause execution to resume.)

EXAMPLE:

```
RETURN          // Must be added after end of section commands
RPT.KEYPRESS:
    STRING PAUSEKEY 1
    INPUT "PRESS <ESC> TO ABORT, ANY OTHER KEY TO CONTINUE" PAUSEKEY
[KEY.ESCAPE] RETURN END.OF.REPORT
RETURN RPT.LOOP
```

This procedure should be located before the REPORTEND command, and immediately after the end of the last group of section commands. The RETURN command shown above on the first line must be added explicitly in the configuration in order to insert this procedure.

If the predefined procedure ENDOFREPORT is used instead of END.OF.REPORT, totalling and other terminating procedures would be skipped. Use of this procedure would be appropriate to use in case of printer misfeed, ribbon breakage, or a similar mishap.

PAGE BREAKS

The REPORT macro automatically maintains the number of lines remaining on the current page in memory at all times while outputting. It will handle things like number of lines required by a SUBTOTAL or BODY image automatically, and if allowed to operate in its default mode, it will reprint headings and subheadings as required at the top of each new page. If you are doing "fancy" paging that requires direct control, however, you will have to provide commands within the applicable section(s) (typically the BODY). The PAGECHECK command checks whether the number of lines given in the command argument remains on the current page. The FORMFEED command sends a "raw" formfeed (that is, it does not print headings at the top of the new page). The sequence [PAGEBREAK] GOSUB NEWPAGE utilizes the PAGEBREAK predefined indicator and the NEWPAGE predefined procedure to print headings at the top of the new page. The following command, for example, would eject the current page and print headings at the top of the next page if there were less than 6 lines left on the current page:

```
        PAGECHECK 6
        [PAGEBREAK] GOSUB NEWPAGE
```

The following command would do the same thing unconditionally:

```
        FORMFEED
        GOSUB NEWPAGE
```

REPETITIVE REPORTS

A report can be put in a loop to execute multiple times for different selection criteria. Put a label at the beginning of the report and then, at the end of the report, CLEAR the report data file and GOTO the label at the begining of the report. Remember to provide some means for the operator to exit the loop.

MULTIPLE COPIES:

To make multiple copies of a report you must: (1) set your OUTFILE to a disk file; and (2) print that file multiple times.

```
        OUTFILE "TEMP.$$$"           // Output to temporary file
        REPORT X BY INDEX.1          // Do report
        ---- report ---
        REPORTEND
// The following lines print the disk file multiple times
        DIRECT_OUTPUT "LST:"         // Output to printer
        INTEGER COUNT
        STRING STRVAR 132
// You must have set up NUMBER OF_COPIES previously.
        FOR COUNT FROM 1 TO NUMBER OF_COPIES
                DIRECT_INPUT "TEMP.$$$"
[NOT SEQEOF]    REPEAT
                    READLN STRVAR    // Reads a line from TEMP.$$$
                    WRITELN STRVAR   // Writes a line to the printer
[NOT SEQEOF]    LOOP                 // End of REPEAT loop
                FORMFEED
        LOOP                         // End of FOR loop
        CLOSE_INPUT                  // Close before delete
        ERASEFILE "TEMP.$$$"         // File no longer needed
```

THINGS TO REMEMBER ABOUT THE REPORT MACRO

1. The REPORT macro may be used only once within a configuration.

2. The maximum line length which can be output is 255 characters.

3. Fields used as subtotal breakpoints must be indexed. They can be in the main file or in any other RELATEd file.

4. Subtotal and total registers are updated only by PRINT commands; MOVE, CALCULATE and DISPLAY do not update the registers.

5. Data should be processed only from database elements or in-memory variables. The contents of formatted windows cannot reliably be used for calculations or other processing.

6. An invisible AUTOPAGE command is executed at the beginning of each command section, and an invisible RETURN command at the end.

7. The BODY section of the report is executed once for every record which satisfies the selection criteria.

REPORTING WITHOUT REPORT

The usual way of configuring reports is by use of the Report macro, often starting with the QUERY program to generate the configuration. The Report macro, however, is by no means the only way of configuring reports--it is merely a pre-programmed convenience suitable to a wide range of purposes. The following configuration outputs a data file (called SAMPLE in this example) to the printer in record number sequence, using a format image with a pagename of IMAGE. Note that it lacks headings, totals and subtotals commonly found in reports produced by the Report macro.

```
/IMAGE

______________________  _______________  __  _____
/*
OUTFILE "LST:"                    //Direct output to the printer
OPEN SAMPLE                       //Open the data file to output
CLEAR SAMPLE                      //Clear the record buffer
CLEARSCREEN                       //Clear the screen
REPEAT
        FIND GT SAMPLE BY RECNUM  //Find the next record by number
        [FOUND] BEGIN
                PRINT SAMPLE.NAME TO IMAGE.1   //Must state TO
                PRINT SAMPLE.CITY TO IMAGE.2   //Window.# since
                PRINT SAMPLE.STATE TO IMAGE.3  //AUTOPAGE command
                PRINT SAMPLE.ZIP TO IMAGE.4    //not used
                OUTPUT IMAGE
        END
[NOT SEQEOF] LOOP                 //Stop at end of file
ABORT
```

THIS PAGE INTENTIONALLY LEFT BLANK

FILE DEFINITION (FILEDEF)

OVERVIEW

The FILEDEF utility enables you to create a complete file definition for data base files used in a DataFlex application. The file definition, once created, contains information about the file name, indexed fields, and field information. Each DataFlex file must have certain information about the structure of the data, file name, indexes etc., established before it can be used. Once done, this information is available, through the data base manager, to any DATAFLEX program. During execution, the programs can adjust their operation to the type and length of the files and fields in use. This gives DataFlex its unique APPLICATION INDEPENDENCE.

APPLICATION INDEPENDENCE means that the same data file can be used in multiple applications without the duplication of effort and storage found in most conventional application programs.

EXAMPLE:

We may want to implement accounts receivable, accounts payable and a mailing list. Instead of having three files with name and address information, we could have one name/address file which is referenced by all three applications. Only this one file need be defined, stored and updated for names and addresses, thus saving both time and storage... a real bargain!

Because these data base files potentially are so integrated with the total application picture, thought and care should be put into the contents and eventual relationships of the files. A well-designed file structure will almost automatically mean an effective and successful application.

STEP ONE - PLAN THE RELATIONSHIPS AMONG THE DATA FILES

If your DataFlex applications uses only one independent file, there are no inter-file relationships. To gain the benefits of the inter-file relational DBMS structure, read the following section carefully to see how to build a multi-file structure. A thorough understanding of DataFlex configuration at the single file level will be extremely helpful in rapidly gaining an understanding of the more complex, multi-file structures. The following section treats data theory generally, not just as it pertains to DataFlex (although it definitely is irrelevant to less-capable software products). There are a number of good books available on this subject, and your mastery of this area may be enhanced by referring to some of them.

WHAT FILE RELATIONSHIPS ARE FOR

File relationships are an advanced concept in database management, but its benefits probably apply to most serious database applications. Although some software products offer the capability, none provides it with the ease and power of DataFlex. File relationships are typically invisible to an operator running a DataFlex application, which is the best way for it to be. Most existing microcomputer applications are designed to use only one data file, but many of those would benefit greatly from a multi-file design approach. Compared with single-file design approaches, file relationships offer the following advantages:

a. More efficient utilization of disk space.

b. Equal data support with a smaller number of records.

c. Greater consistency of data in groups of records having data elements in common.

d. Improved efficiency in changing data applying to groups of records having the data in common.

To illustrate the concept, let us use a simple mailing list of indivuals at their business addresses. Each record in the list would have the individual's name, name of the company where they work, address of the company, and a phone number. So far, this would seem to be a typical single-file application. But suppose that groups of fifteen or twenty of these people all worked at the same company. You would still want an individual record on each person, but if one of the companies moved, you would have to find each record for someone who worked at that company and change the address in each one. If, on the other hand, a relational approach were taken, company names and addresses would be kept in a separate file, and each person's file would have in it his company's name, but no address. The company name (to support the relationship) and address would be in the company file to which each person's file related. There would be many fewer company files than person files, and each person's record would be smaller (having no address or phone number in it). Moreover, when the ABC company moved, the addresses of all 27 people on your list who work there would be changed (consistently in every record) by changing only the company record for the ABC company. And there would be nothing apparently missing from each person's record, either. DataFlex can be configured to show each person's full company name and address as though it were part of the person's actual record. This process is invisible to the operator, unless explicit provisions are made to render it visible (which wouldn't normally be desirable). And when John Doe moved from the ABC Company to the XYZ company, changing his whole address would be as easy as changing his company's name from ABC to XYZ (provided there was already a record for the XYZ Company in your company file--if there wasn't, you could create a new one, right there "in" John Doe's record entry screen).

But it goes even further. DataFlex can relate **ten** files to each other online simultaneously (in 16-bit computers--capacity in 8 bits is five). If we were selling to these individuals, we could keep all our sales in a file **related to** our persons file by the person's name. Then, when looking at a sale, we could view the customer's complete name and address (now looking through **two** files, all the way to the related company record) with each sale, even though only the customer's name (or account number) is part of the sale record.

These, then, are the hallmarks of the need for related files: groups of records having data elements which must be the same at all times (like customer for groups of sales transactions), or, looking in the opposite direction, series of identically-structured data groups in a record. DataFlex provides a **heirarchical**, or one-to-many type of file relationship which can improve the speed and efficiency of your application greatly.

GENERAL LAYOUT OF FILES

Drawing a schematic of your file relationships as a "tree" structure is the first step in configuring the application. This exercise gives you a chance to establish not only the files involved in the application but also gives the ability to see how data can be grouped together effectively and specifically to each of the files.

EXAMPLE:

```
                 DATA FILE RELATIONSHIPS "TREE"

                  CUSTOMER          VENDOR
                    FILE             FILE
                     ^                ^
                     |                |
                     |      ------> PARTS
                     |      |        FILE
                     |      |         ^
                     |      |         |
                     |      |         |
                     |      |         |
                   SALES----'     PURCHASE
                TRANSACTIONS    TRANSACTIONS
                    FILE            FILE
```

DataFlex utilizes "Data Relationships" to maintain the inter-file relationships you have defined. The lines connecting the files show the relationships among the files. In the actual file definition these relationships are established through the use of related fields. A representation like this one will make the RELATIONSHIPS among the files readily apparent. The fields through which different files are related must be of the same type (ASCII, numeric or date) and the same **length**.

The relationships in this representation point UP the tree. There are four file relationships in the illustrated structure:

1) SALES TRANSACTIONS TO PARTS
2) PURCHASE TRANSACTIONS TO PARTS
3) SALES TRANSACTIONS TO CUSTOMER
4) PARTS TO VENDOR

When visualizing files as a tree structure, remember that the topmost files or "leaves" of the tree are those that do not draw on other files for information (e.g., the VENDOR file above has no PARTs in it, but every record in the PARTS file has a vendor in it). Lower down on the tree there are files (like PARTS) which depend on the VENDOR file for information. The bottom of the tree is the TRANSACTION files where data from all of the other files contributes to existing and new records (TRANSACTIONS).

In DataFlex, the lower files on the tree are RELATED TO higher files. This means that an inventory PARTS record which is related to a VENDOR record can automatically access (display or print) information in that VENDOR record.

In relational multi-file structures, one element of data could have a field entry in several levels of the tree (i.e., VENDOR ID could be part of the VENDOR, the PARTS and the TRANSACTION files). Like fields in different files are logically "related" together by DataFlex. In this manual, and in the prompting you will encounter in the configuration utilities, the term "relates" or "relationship" will be used frequently. It is used to refer to this logical linking of fields in different files which is controlled by DataFlex's Data Base Manager.

The relationships point UP the tree; the PART ID in the TRANSACTION file relates TO the PART ID in the PARTS file and the VENDOR ID in the PARTS file relates TO the VENDOR ID in the VENDOR file. Therefore, transactions can automatically include not only PARTS data, but PARTS AND VENDOR data. DataFlex can automatically find records up and down the tree from these relationships.

HOW TO CREATE RELATED FIELDS

There are two criteria by which to choose the field(s) through which to relate two files: (1) for the Data Base Manager to find a record in one file from a record in another file (i.e., find the VENDOR record from a PARTS record); and (2) to have the system automatically carry the value in a field forward to another file for indexing or report selection.

THE RELATED FIELDS IN DIFFERENT FILES MUST
BE OF THE SAME TYPE AND LENGTH!

When a field relationship is created for FINDING purposes, the field being related TO **must uniquely identify the record.** If the relating field isn't the higher record's record number, then there should be an index made up of that field alone. (Record numbers are inherently indexed, and inherently unique, which is why they are the prime choice for relating fields.) The lowest-numbered relating field in the file definition is the one which provides the primary relationship. This is referred to as the **PRIMARY relationship.** (Again, the record number is always the lowest-numbered field in any file definition.

When a record that has relationships to other records is FOUND, the records it relates TO will be found automatically. This makes the records related TO appear as though they were part of the original record. The use of multiple files is typically invisible to the operator).

The related-to file's record number is normally the best choice of field on which to base a primary relationship. Relating by record number has three advantages: it is fast (since no index access is required), the record number is ALWAYS unique, and it cannot be changed.

A field in one file that relates to a record number in another should be a numeric field of 1, 2 or 3 bytes' length. The length is determined by the maximum potential number of records in the file related TO. Remember, DataFlex stores two digits per byte of field length, so a one byte numeric field will store related record numbers up to 99, 2 bytes up to 9999, etc. Three bytes is always a safe length. This field, and any other related field in a new record, will be updated by the field it relates TO when the record is saved.

When you are using a record number as the primary relationship between the records of different files, you may still wish to have other relationships between the records for more convenient multi-file FINDing. Remember that these secondary relationships must be based on field numbers that are higher than the numbers of the fields on which the primary relationship is based.

In the typical case, there would be many records in the file being related FROM relating TO each record in the file related TO (each VENDOR would supply many PARTS). Wihtout this relationship, the field being related FROM would NOT uniquely identify a record (you can't find an individual PART _just_ by knowing the VENDOR). Therefore, the PARTS file would be indexed by VENDOR for tasks like listing all the parts for a vendor AND something else, like part ID, to facilitate the finding of an individual part irrespective of who its vendor might be.

Decide at this point which fields (Record Number, Name, Part Number, etc.) will be used for RELATIONSHIPS in your application. Below are four of the five files in the schematic shown two pages ago. This rendition, however, displays the actual fields in each file, and the arrows show which fields the relationships are based upon. Hint: one of the relationships shown is a secondary relationship. Can you find it?

EXAMPLE OF FILE STRUCTURES AND RELATIONSHIPS:

```
                                                  SALES
VENDOR FILE               PARTS FILE              TRANSACTIONS FILE
-----------               ----------              -----------------
:                         0  RECORD NUMBER        0  RECORD NUMBER
0  RECORD NUMBER <-------1  VENDOR RECORD #      1  DATE
1* VENDOR ID <-----------2* VENDOR ID            2  QUANTITY
2  ADDRESS1               3* PART ID <---------3* PART ID
3  ADDRESS2               4  QTY IN STK           :
                          5  PRICE                :
                          6  DESCRIPTION          :
                                                  :
                                                  :
CUSTOMER FILE                                     :
-------------                                     :
0  RECORD NUMBER <--------------------------------4* CUSTOMER RECORD #
1* CUSTOMER NAME                                  5  AMOUNT
2  ADDRESS1
3  ADDRESS2
4  CREDIT LIMIT
5  TOTAL SALES
```

The arrows (<---) represent a relationship and the "*" represents a field which is indexed. Note that other relationships could exist among these files, and/or to other files, and other fields might be indexed for finding and reporting purposes.

VENDOR RECORD # in the PARTS file is used for the PRIMARY relationship with the VENDOR file to speed up the RELATE process since no indexing is needed with Field 0, the record number. The VENDOR ID is also provided as a secondary relationship to allow reporting and FINDing by VENDOR ID in the PARTS file.

The VENDOR ID field **must** be of the same length and type in both the VENDOR and PARTS files.

The PART ID in the PARTS file and in the SALES TRANSACTIONS file must be of the same type and length in both files. PART ID in the PARTS file must be indexed as the only segment in an index since it is used as the primary relates-to field from the SALES TRANSACTION file.

The PART ID in the SALES TRANSACTIONS file is listed as indexed, but if we did not want part ID to be included in the SALES TRANSACTION file, we could have made its Field 4 the smaller and faster PART RECORD #, and based the relationship on that field instead.

Reports of transactions could be printed either by PART ID or CUSTOMER record number, since both indexes are present in the SALES TRANSACTIONS FILE.

STEP TWO - LIST THE FIELDS

Determine the specifications of the data fields that will be required in each file, including their type, length and, for numeric fields, the number of places to the right of the decimal point. A convenient way to do this is to create the screen IMAGE of the data entry form for each file. (Refer to the section of this manual on "Formatting With Images".) Then let FILEDEF's Option 7 read this IMAGE and create a preliminary file definition automatically! Option 7 will prompt you through the process, as described below in Step 4. If this file will relate to other files, remember to be sure that the fields used for the relationships have the same length and type in each of the related files.

EXAMPLE:

```
/VEND
.............................................................
:                                                           :
:                      VENDOR RECORD                        :
:                                                           :
:          VENDOR ID: ________                              :
:                                                           :
:       COMPANY NAME: ______________________________        :
:            ADDRESS: ______________________________        :
:               CITY: _______________ ST: __ ZIP: _____     :
:                                                           :
:       CREDIT LIMIT: $________.__  TOTAL CREDIT: $________.__ :
:                                                           :
:         ENTRY DATE: __/__/__                              :
:...........................................................:
/*
```

This image could later be used directly by FILEDEF to create a definition. Note that the image is terminated by a "/*" character combination. This is an "end of image" marker and is necessary for DataFlex to distinguish the image from the configuration commands.

After Option 7 has run, FILEDEF's Option 3 would produce a chart like this:

```
FIELD  FIELD   FIELD    FIELD    DEC   MAIN   RELATES--TO
NMBR   OFFSET  LEN      TYPE     PTS   INDEX  FILE  FIELD
-----  ------  -----    -----    ---   -----  -----------
  1       1      8    ASCII              0      0     0    VENDOR ID
  2       9     30    ASCII              0      0     0    COMPANY_NAME
  3      39     30    ASCII              0      0     0    ADDRESS_LINE_1
  4      69     30    ASCII              0      0     0    CITY
  5      99      2    ASCII              0      0     0    STATE
  6     101      5    ASCII              0      0     0    ZIP CODE
  7     106      5    NUMERIC     2      0      0     0    CREDIT_LIMIT
  8     111      5    NUMERIC     2      0      0     0    TOTAL_CREDIT
  9     116      3    DATE               0      0     0    ENTRY_DATE
```

OVERLAPPING FIELDS (FOR EXPERTS ONLY)

There may be times when you need to have more than four field segments in an index (four is the segment limit for an index in 8 bits; six is the limit in 16 bits). See the section of this manual on "Designing Databases" for a full discussion of single- and multi-segment indexes and indexing. For these special cases, DataFlex has another field type called the overlapping field. This field type enables the user to address a group of fields as though they were one large field. The overlap field spans other fields already existing in the file definition. The fields spanned by the overlap **must** occur consecutively in the file definition.

To use this feature, first append a field to the file of the type OVERLAP. Then set the FIELD OFFSET of the overlapping field to the field offset position of the first sub-field and the FIELD LENGTH of the overlapping field to the total length of all of the sub-fields you wish to include in the index. Then, of course, specify the index number for the index based on the **overlapping** field (rather than on the fields it is composed of).

This is what the chart on the preceding page would look like if an overlap field comprised on the first two fields were added:

FIELD NMBR	FIELD OFFSET	FIELD LEN	FIELD TYPE	DEC PTS	MAIN INDEX	RELATES--TO FILE	RELATES--TO FIELD	
1	1	8	ASCII		0	0	0	VENDOR_ID
2	9	30	ASCII		0	0	0	COMPANY_NAME
3	39	30	ASCII		0	0	0	ADDRESS_LINE_1
4	69	30	ASCII		0	0	0	CITY
5	99	2	ASCII		0	0	0	STATE
6	101	5	ASCII		0	0	0	ZIP CODE
7	106	5	NUMERIC	2	0	0	0	CREDIT_LIMIT
8	111	5	NUMERIC	2	0	0	0	TOTAL_CREDIT
9	116	3	DATE		0	0	0	ENTRY_DATE
10	1	38	**OVERLAP**		0	0	0	

STEP THREE - DETERMINE THE KEY FIELDS

DataFlex's indexes allow you to FIND records by fields which are defined as indexed. The fields you wish to index should be determined at this point in the FILEDEF operation. Each DataFlex index can be composed of up to 4 fields on 8 bit systems, 6 fields on 16 bit systems. There can be up to 4 indexes for each file (9 on 16 bit systems). Each index defined will be stored in an index file that has the same rootname as the data file. It will be opened automatically when the data file is opened.

Keep in mind that each index requires disk storage and a small amount of update time when saving or deleting records. System size and performance may become a consideration in the design of your key file structure. A rule of thumb is that an index will take about 2 times the amount of disk space as the number of bytes in the data field(s) that it is indexing. For example, if an index is made up of 1000 ten byte part numbers, the index file size will be about 20K. All indexes for a database are kept up-to-date on-line automatically.

A field does NOT have to be indexed for it to be used to SELECT records from the database during output operations. For example, if a STATE field in a mailing list record was not indexed, you could still make a report of all records matching a particular STATE value. If you did, DataFlex would have to evaluate the STATE field of every record. For this reason, indexing often does speed up the selection process.

If a field in one file is being related to by records in another, it must be the only field in the index. This is required for the relate process to uniquely identify a related record. Overlapping fields can be used here to create a single-element index for relating to, which is actually made up of data from several fields. If this is done, the relating file must also have an overlapping field of the same length to support its end of the relationship.

DUPLICATE RECORDS

If you wish to allow multiple records in a file to have the same key value, include field zero (the record number) as the last segment of the index. Since record numbers are always unique, their inclusion with other index elements can make otherwise duplicate key values unique. Note that a field CAN NOT be indexed with record number AND be the target of a relationship.

The record number, which is addressed as field zero in DataFlex, can also be considered INDEX ZERO. This creates an often useful access method (e.g. transactions which need to be accessed by order of entry) which has no disk storage overhead. There is actually no INDEX ZERO (no file called FILENAME.K0); this is DataFlex's method of defining that access is to be made by record number. You may always FIND or report by record number. A file may exist with no other indexes. The record number for the record currently in the buffer is maintained under as filename.RECNUM.

Where a field participates in more than one index (perfectly acceptable), one of the indexes must be designated during the definition as the MAIN index for the field for FINDing operations. The MAIN index will be the one automatically used by DataFlex to FIND records from the file when a given field is selected as the source of a key value for record retrieval.

EXAMPLE:

In the example file structure above we might have three indexes:

1) VENDOR ID (unique).
2) COMPANY NAME plus RECORD NUMBER (duplicate company names OK).
3) CREDIT LIMIT plus VENDOR ID (duplicate credit limits OK).

Number 1, VENDOR ID, would generally be used to support relationships, while the others would be useful for FINDing records for maintenance and reporting.

Notice that VENDOR ID participates in two indexes. This is fine, but one must be selected as the MAIN index for that field. In this case, Index One, where VENDOR ID is the first segment, is designated.

To allow for duplicate company names in Index Two, Field Zero (the record number) is included in the index to create unique combinations.

STEP FOUR - USE FILEDEF TO PUT IT ALL TOGETHER

This section takes a procedural step-by-step look at how DataFlex's FILEDEF utility can be used to create and maintain the data files of an application.

To execute the program, type "FILEDEF" at the operating system "A>" prompt or select it from the CONFIGURATION MENU. The program will sign on and present a series of prompts in order to obtain the data needed to define the data file. Each of the prompts from FILEDEF is presented here with the how to's on their use and their relationship to the rest of the DataFlex system.

IF YOU MAKE AN INCORRECT ENTRY:

Some FILEDEF menu choices proceed through a series of prompts, which must be answered in order to continue. If at any point, you are not sure of the answer, or you key a wrong response, it is still necessary to answer all of the questions, at least with something that satisfies the program, in order to proceed and get back to the FILEDEF operations menu. Once all the questions have been answered, however, the menu (see below) gives you the opportunity to modify your earlier answers or to abandon them altogether (choice 10), and leave your original file (if any) unchanged.

ENTER FILE NUMBER:

When run, the FILEDEF Program prompts the configurator with a list of currently defined files. You may modify one of these, enter an unused file number to start a new file or reactivate one not currently on the list.

As a shortcut, you may enter FILEDEF <FILENUMBER> at the operating system command level to pass a file number directly to the program without going through the files menu.

NEW FILE (Y/N)

If you have selected a file number that does not appear in the file list then you must tell FILEDEF if this is a new file or an inactive one.

If you are creating this file for the first time answer the question with a "Y".

If a file definition for this file already exists, answer the question with the "N" response (you can just press ENTER to default to the "N" response), which will give you the chance to reactivate the file. The system will then ask for the ROOT name, USER DISPLAY name, and Data-Flex filename of the file.

DataFlex application configurations purchased separately which are installed on an existing DataFlex system must have their file numbers re-activated as described above.

If you are modifying or re-activating an existing file definition, the FILEDEF menu will be displayed. Otherwise, for a new file, some preliminary information will be required:

ROOTNAME

The ROOTNAME refers to the base name of any DataFlex data or configuration file created by FILEDEF. The ROOTNAME must conform to operating system filename conventions of up to eight letters and numbers. DataFlex will append the appropriate extension based on the use of a particular physical file (see the appendix). Make up an 8 character or smaller rootname for your file WITHOUT AN EXTENSION!

If you wish to specify the drive on which DataFlex is to place the data files, prefix the file name with the drive specification in the standard format: drive letter (A..P) followed by ":" followed by filename.

EXAMPLE ROOTNAME W/DRIVE SPEC.: B:MYFILE

This would be a file called "MYFILE" located on disk drive "B".

USER DISPLAY NAME:

The User display name is the one which will be displayed in the FILE SELECTION menus by DataFlex programs. Enter the name you would like to see displayed to the operator for this file. If the file name is preceded with a "@" it will not display on the runtime file display menus. The maximum length of the user display name is 30 characters.

DataFlex FILENAME

The DataFlex filename can be up to ten characters long and must start with a letter. The DataFlex filename is used to reference a data file from configurations. In that it does not contain drive specifications it is a "soft" reference to a file that makes an application independent of a particular hardware configuration. It is also used, along with a field's Tagname, to define DATA BASE ELEMENT names within a file.

RECORD LENGTH

This is the physical size (in bytes) of the record you will be creating. DataFlex will automatically block small records into 512 byte blocks. Larger records can be any multiple of 128 bytes. You can assign the record length by selecting option 4 from the FILEDEF operations menu. If there is a conflict between what you assign as the record length and what the system needs to correctly block the record, the necessary record length will be blocked automatically by FILEDEF when you enter the record length.

If you know the record length at this point, you may enter it. If you are creating a definition from an IMAGE and don't know the record size, you may initialize it to zero. The READ SCREEN option will then set the record length for you. DataFlex will block records in all sizes from 1 to 26 bytes, 28 bytes, 30, 32, 35, 39, 42, 46, 51, 56, 64, 73, 85, 102, 128, 170, 256, and upward in multiples of 128 bytes, to a maximum to 4 kilobytes.

MAX # OF RECORDS

The value entered here is the maximum number of records to be stored in the database for this file. The figure entered here is non-critical unless you elected the system option (see Installing DataFlex) which provides for pre-allocation of disk space for the maxium possible size of your database. Otherwise, you may feel free to make some growing room available for the number of records in a particular file.

The absolute maximum number of records in a DataFlex file is 64,000. NOTE: If the need arises to expand the maximum number of records after a file contains data, re-enter FILEDEF and enter a larger value for this field. Then, run the REINDEX utility to rebuild the index(s) under the new parameter.

REUSE DELETED SPACE (Y/N)

For most applications you will want DataFlex to reassign the disk space released when a record is deleted from a file, however, in applications where you want to use the record number as an audit trail or as a transaction number, this would not be desirable. To allow the re-use of deleted space, enter <Y>es; otherwise, enter <N>o.

MULTI-USER RE-READ (Y/N):

If you are running on a multi-user system (and you have a multi-user version of DataFlex) and you wish for this file to utilize the feature of allowing multiple users to write to the same record at the same time then enter a 'Y', otherwise, enter 'N'. NOTE: MULTIUSER option of SETSCREEN must be set to (Y) to utilize RE-READ.

THE FILE DEFINITION MENU

After completing the entries just covered about ROOTNAME, etc., the FILEDEF operations menu will be displayed. It looks like this:

```
 1: CREATE/EDIT FIELD SPECIFICATIONS
 2: CREATE/EDIT INDEXES
 3: DISPLAY/PRINT FILE DEFINITION
 4: SET FILE PARAMETERS AND NAMES
 5: ERASE DATA FILE
 6: SET FILE INACTIVE
 7: CREATE FILE DEFINITION FROM SCREEN IMAGE
 8: CREATE FILE DEFINITION FROM .DEF IMAGE
 9: SAVE DEFINITION AND EXIT FILEDEF
10: ABORT FILEDEF, DO NOT UPDATE DEFINITION FILE
```

You may enter one of the option numbers (1 - 10) to select your next action. For new files you MUST either execute Options 1 and 2, or Option 7. The results of using Option 1 or Option 7 are the same: creation of a DataFlex file definition. The methods used to reach that end are different: option one takes you field by field through the record definition where you make the specification entries manually. Option 7 functions just like the DataFlex AUTODEF Utility and determines the record structure automatically from the screen image.

OPTION 1 - CREATE/EDIT FIELDS

This option allows you to create new field attributes or change existing ones in a record definition.

If the file which is currently active in FILEDEF is not a new file, then all entries under this option will contain a default to the existing value for each field that is prompted (except for any new fields you may be adding).

The prompt will be displayed with the existing value in darts: <DEFAULT>. If the value in the darts is to be maintained as the current response, then hit <RETURN> to skip the question and maintain the value; if the default is not correct then enter the new information, followed by <RETURN>.

Each field in the record will be presented sequentially for modification unless you enter a field number to jump to or "E" (exit) on the verification step (SEE BELOW).

When you select Option 1 for the FILEDEF operations menu, you will be positioned at Field 1. The current field specification will display with the field edit menu. The field edit menu looks like this:

```
        OPTIONS:
            G   -   Go to field number
            I   -   Insert before this field
            D   -   Delete this field
            A   -   Append a field to end.
            C   -   Change this field
            R   -   Reposition field offsets
            +   -   Display next field
            -   -   display previous field
            E   -   Exit to master menu

        ENTER CHOICE:
```

FIELD EDIT OPTIONS:

G - Go to field number - Repositions you at the field number entered and displays the current definition of that field. You will be prompted for the field number to go to after entering the <G> option.

I - Insert before this field - Creates room for a new field by incrementing all the higher field numbers by one. Then, the field specifications for a new field may be entered under the newly-available field number.

D - Delete this field - Deletes the current field and reduces all the higher field numbers by one to adjust for the deletion.

A - Append a field to end - A new field will be created as the last field in the File Definition. You must enter the new field's specifications (see below).

C - **Change or edit this field** - Allows you to re-enter the specifications for the current field.

R - **Reposition field offsets** - Required to reset field positions after the file definition has been modified. The positions of the fields in the record buffer (field offset) must be recalculated whenever any field is added, deleted, or changed in length.

+ - Moves you forward one field.

- - Moves you back one field.

E - **Exit to master menu** - Returns you to the FILEDEF menu.

<RETURN> - Always moves editing to the next field. For example, if you are on Field 2 and press <RETURN>, Field 3 will be displayed. <RETURN> will create a new field if you are currently on the last field in a file definition.

You will be asked the following questions for each field:

****** FIELD # ******
# is the number of the field you are currently editing.

FIELD **NAME: (tagname)**
The tagname you enter here will be utilized by FILEDEF and QUERY for prompting and documentation, and will be displayed in the file definition. It will be used, along with the DataFlex filename as part of the Data Base Element name, in the DataFlex Command Language. The tagnames are stored for each file in a separate ASCII file called rootname.TAG. The maximum tagname length is 15 characters. Valid characters in tagnames are:

```
[ 'A...Z', '0...9', '@', '_', '#' ]
```

The **first** character must be alpha (A...Z). Spaces and slashes are invalid, and will be replaced with an underscore (_).

FIELD TYPE: (A=ASCII, N=NUMERIC, D=DATE, O=OVERLAP)
This will set the field to either:

```
ASCII (numbers and letters)
NUMERIC - (numbers only)
DATE - Stored Julian; output format: MM/DD/YY
OVERLAP - A field which re-defines areas of the
          record.
```

The next prompt depends on the field type entered above.

For ASCII fields:

NUMBER OF CHARACTERS
This is the maximum number of characters to be assigned for this field.

For NUMERIC fields:

NUMBER OF CHARACTERS TO THE LEFT OF THE DECIMAL POINT
Each number is broken into two parts: left and right of the decimal point. Enter the number of characters to the left of the decimal point. (Minimum of 1, maximum of 14.)

NUMBER OF CHARACTERS TO THE RIGHT OF THE DECIMAL POINT
Enter the number of characters to the right of the decimal point (0 to 4). The characters to the left and right of the decimal point are added together by FILEDEF to compute the total field length. For DATE fields, no questions are asked. Dates are ALWAYS stored as three-byte numbers. (

For OVERLAP fields:

FIELD OFFSET:	OFFSET OF FIELD IN RECORD This number represents the relative number of bytes from the beginning of the record to where the current overlap field starts. If there are no previous overlap fields in the record, then this number will be the number of bytes in the record up to the first overlapped field. If there are previous overlap fields, their combined length must not be counted in developing this number.
FIELD LENGTH:	LENGTH OF FIELD IN BYTES This is the number of bytes the overlap field will span in the record. It should be the total length of the overlapped fields.

For ALL field types:

RELATIONSHIP TO FILE:	FILE NUMBER OF RELATED FIELD If this field relates TO a field in another file, enter the number of that file here. A printout of the file definition of the related-to file would be helpful for answering this and the next question, if there is a relation. If there is no relationship FROM this field, enter a zero.
RELATIONSHIP TO FIELD:	(FIELD NUMBER OF RELATED FIELD) If the question above was answered in the affirmative (with a number), then enter the number of the field which is being related TO. Otherwise, enter a zero.

The creation or editing of the selected field is now complete. The information for this field will be re-displayed for inspection purposes. If you wish to make further changes, enter a 'C'. If all is correct, enter a "+" to move on to the next field, or enter an 'E' to return to the activities menu.

OPTION 2 - CREATE/EDIT INDEXES

Option 2 allows you to create or change the index(s) for a file. The following "dialog" with FILEDEF allows you to easily define how you want to your file indexed:

FILEDEF PROMPT	YOUR RESPONSE
HOW MANY INDEXES FOR THIS FILE:	A NUMBER IN THE RANGE FROM 1 TO 4 (One to nine in 16-bit computers) This will correspond to the number of unique indexes you wish to have automatically maintained for this file.
-- THE FOLLOWING WILL REPEAT FOR EACH INDEX --	
****** INDEX # ******	-- NONE -- The number of the index you are entering will be displayed.
HOW MANY FIELD SEGMENTS IN INDEX:	# OF FIELDS USED IN INDEX, RANGE: 1 - 4 (6 on 16 bit) Each index may have up to six segments in 16-bit computers. For example, you might want to find on a field named LASTNAME, but also include the FIRSTNAME and MIDLINIT fields in the index (for a total of 3) in order to have records with the same last names (Smith, Robert; Smith, Samuel) properly sequenced by first name and middle initial within the list of Smiths. Enter the number of fields to be included in this index.
ON LINE INDEX (Y OR N)	If you answer Y(es), the index will be maintained on line and updated every time a change is made to the database. If you answer N(o), the index will function normally only passively as part of the file definition, and will be updated only when specifically processed through REINDEX as an "ad hoc" index. Reasons for answering N(o) include: (a) your system's capacity for maintaining on line indexes is already consumed by more important indexes; (b) this index is needed only at infrequent intervals for a periodic report, and not for routine FINDing; (c) this index is large and complex, and on line updating in a large database would slow processing time down more than is justified.

-- THE FOLLOWING WILL REPEAT FOR EACH SEGMENT --

SEG X FIELD NUMBER:

FIELD NUMBER OF INDEX SEGMENT. Enter the field number you want for this index Segment. The order in which you enter each field number will determine its indexing precedence.

-- THE FOLLOWING APPEARS ONLY FOR FIELDS ALREADY IN ANOTHER INDEX --

WHICH INDEX, (X OR Y), SHOULD BE USED WHEN FINDING BY THIS FIELD?

If the field specified above is used in another index already, either this (the current--X) index or the earlier one should be designated for FINDing from a screen with the cursor in the window for this field. Generally, the index in which this field is first, or nearest the first, should be chosen. Your choice will be designated the "main" index for the current field in the File Definition (see printout below).

INITIALIZE INDEX (Y OR N):

If any of the above information was changed from previous (default) data for an existing index, then the index will automatically be initialized. However if no changes have been made, you will have the option to initialize or not. If an index is initialized and there is data present in the data file a message will appear telling you to "REINDEX" the index. (See UTILITIES section, REINDEX).

DELETING AN INDEX

If at any time you wish to delete an index, select Option 2 from the FILEDEF operations menu. Then enter the OLD number of indexes in responding to the prompt for number of indexes and press <RETURN>. If you want to delete Index 1, enter zero for its number of segments. For a higher-numbered index, press <RETURN> repeatedly to default through the questions until you have reached the prompt for the index you wish to delete. Then enter a 0 for that index's number of segments. The index numbers for indexes following the deleted one will remain the same ... i.e. if index 2 was deleted from a 3 index file, indexes 1 and 3 would remain. When you delete an index, its file will be deleted from the disk.

OPTION 3 - PRINT FILE DEFINITION

The third option on the menu, PRINT FILE DEFINITION, will allow you to inspect the file that has been defined.

You have the choice of sending the output to either the screen (default), the printer, or a file. If you send the output to the printer make sure that it is turned on, and set ON LINE before entering this command to avoid a lockup.

If you are sending the output to a file, the filename will be the ROOTNAME that you have supplied for the file that you are editing, and it will have an extension ".DEF" (or rootname.DEF). Sending the output to a file can help you document the application. After sending the output to the file, you can edit it with your text editor and/or include it in other documentation. Refer to Option 8 below, "CREATE FILE DEFINITION FROM .DEF FILE" for more information.

A completed file definition, when printed out, looks like this:

```
        FILE DEFINITION LISTING FOR FILE #21
***************************************************
        FILE ROOT NAME    = VENDOR
        USER DISPLAY NAME = VENDOR
        DATAFLEX FILENAME = VENDOR
***************************************************
        RECORD LENGTH = 128 (USED = 112)
        MAX NUMBER OF RECORDS = 400  (USED = 5)
        DELETED SPACE IS REUSED
        MULTI-USER RE-READ INACTIVE
***************************************************
FIELD   FIELD   FIELD     FIELD  DEC  MAIN  RELATES--TO
NMBR    OFFSET  LEN       TYPE   PTS  INDEX FILE  FIELD
-----   ------  -----     -----  ---  ----- -----------
   1        1   30        ASCII         1       0    0   VENDOR_NAME
   2       31   30        ASCII         0       0    0   ADDRESS
   3       61   14        ASCII         0       0    0   CITY
   4       75    2        ASCII         0       0    0   STATE
   5       77    9        ASCII         0       0    0   ZIP
   6       86    3        ASCII         0       0    0   PHONE_AC
   7       89    3        ASCII         0       0    0   PHONE_EXCHG
   8       92    4        ASCII         0       0    0   PHONE_NUMBER
   9       96    2      NUMERIC  0      0       0    0   TERMS
  10       98    4      NUMERIC  1      0       0    0   CREDIT_LIMIT
  11      102    4      NUMERIC  1      0       0    0   CREDIT_USED
  12      106    4      NUMERIC  1      0       0    0   PURCHASES
  13      110    3         DATE         0       0    0   LAST_PURCHASE

INDEX 1:  FIELD SEGMENTS:  <1>    <0>
```

OPTION 4 - SET FILE PARAMETERS/NAMES

Option 4 on the menu, SET FILE PARAMETERS/NAMES, will allow you to change the file names, record length, maximum number of records in a data file, and the REUSE DELETED SPACE OPTION. Defaults are supported here for ease of operation. Refer to the NEW FILE section for information on these questions.

OPTION 5 - ERASE DATA FILE

This option is used when ALL data in a file is to be deleted. This could come in handy when you want to delete sample data, and bring the system up under actual conditions, or when an end of period zeroing of a data file is necessary. This is a POWERFUL command. You will be asked to confirm that you actually do want to zero the file before it executes. This prompt will only respond to a capital "Y" (we want you to be absolutely sure). After the data file has been erased, the remaining file definition, without data, may be saved with Option 9. If this is not done, the file definition will also be erased.

OPTION 6 - SET FILE INACTIVE

This option will allow you to delete a name from the list of files that appears at the start of FILEDEF without actually deleting the file definition or data files from the disk. You will need to confirm that you actually want to perform this action. (Again, with a capital "Y" only). NOTE, to reactivate a file, upon entry into FILEDEF, select the file number of the file that you want to reactivate, and respond to the question "NEW FILE" with "N". The procedure for removing all trace of a file from the system is to first set the file inactive through FILEDEF, then use the ERASE command in the MENU or your operating system.

OPTION 7 - CREATE FILE DEFINITION FROM SCREEN IMAGE

This option reads in an IMAGE (created with a text editor) from a file which you specify in response to its prompt, and all the window information in the image file is used to create a basic file definition and optionally an ENTER configuration. Refer to the section on "Formatting With Images" for more on how to create IMAGEs.

You will be prompted for the file name of the screen IMAGE to read in. If the file is located on a drive other than the default drive, precede the file name with the proper drive specification. If you want an ENTER configuration created for this file, answer "Y" to the next question. The configuration created will have the ROOTNAME of the IMAGE used, with an .FRM extension added. If you want only to create the file definition and not a configuration file, respond with "N" to this prompt.

The screen image will be evaluated from top to bottom, left to right, to extract the types and lengths of the fields in the image. This process will replace any field definition that may already exist under the selected file name. If you require a more complex file definition with related fields or overlapping fields, you may use this option to create the basic field structure and add the needed relationships using Option One.

After using Option 7, define the indexes with Option 2.

OPTION 8 - CREATE FILE DEFINITION FROM .DEF FILE

The ASCII file with filename extension .DEF which can be created with Option 3 (see above) can be used to recreate the structure of the file definition. This option has other uses. Say, for example, that you had a file definition on an 8-bit computer that you wanted to move to a 16-bit model. We will assume that DataFlex is installed on both machines. You can use Option 3 to create an ASCII text file for the file definition. This ASCII file can be transported to the 16-bit machine by a data communications program. Once on the 16-bit machine, this .DEF file can be read by FILEDEF Option 8 to recreate the file definition there. A new file definition created in this manner will be empty of data. Of course this process serves to transport file definitions in the other direction also.

OPTION 9 - SAVE DEFINITION/EXIT FILEDEF

This option saves the file definition just created or edited to the disk. After this option is executed, you will be asked if another file definition is to be loaded for file definition. Respond "Y" if you desire to edit more, or "N" to exit FILEDEF.

If you elected pre-allocation of required disk space for the maximum possible size of database files in installing DataFlex (see Installing DataFlex), the files will be created at this time (if you are establishing a new database) **in their maximum size when completely full of data.** You should make certain enough space is available for the new files. Your screen will display a sequence of dots during the time it is creating the new files, which may consume an hour or more if the new files are sizable.

If another file is to be loaded, you may also respond to the prompt with the next file number to load if it is known. This reduces the time required to load the menu and begin editing on the next file.

OPTION 10 - ABORT FILEDEF

Option 10 will exit FILEDEF without making any changes to the file definition. **NOTE:** If you used Option Five to erase data, and you then abort with this option, the selected erasure will **not** occur. The same is true if you deleted an index or took **any** other action.

THIS PAGE INTENTIONALLY LEFT BLANK

SAMPLE APPLICATIONS

A sample multi-file configuration is provided with DataFlex. It is an inventory / order entry system which you may find quite useful. However, that is not its purpose. The order entry sample is, rather, intended as a teaching tool and as a starting place should you wish to construct such a system.

The order entry configuration as supplied is fully operational including sample data files on your master disks. It is recommended that you become thoroughly familiar with how and why it works BEFORE attempting to create your own multi-file configurations.

The first step in understanding DataFlex is to run the demo configuration. From the Master Menu, you may select "DATAFLEX DEMO" to run the order entry system.

A SINGLE FILE SCREEN...

First we will deal with DataFlex using only ONE file at a time. To demonstrate operation at this level we will use the CUSTOMER file. Although the CUSTOMER file is related TO by other files, it relates to no others, and thus can be used by itself.

THE CUSTOMER SCREEN:

There is only one screen used to access and update the CUSTOMER file. Select "CUSTOMER FILE MAINTENANCE" from the menu. The screen should look like this:

```
DATA ACCESS CORPORATION                              CUSTOMER MASTER FILE
=========================================================================


       CUSTOMER NUMBER: ___.


          NAME: ______________________________

       ADDRESS: ______________________________

          CITY: ______________  ST: __  ZIP: __________


       DISCOUNT: _.%

       PURCHASES/MONTH $______.__  PURCHASES/YEAR $______.__

       AMOUNT DUE  $______.__
```

This is a simple, single file screen configuration to do file maintenance and look-up on the CUSTOMER file. Refer to the Operator's Guide section of this manual on how to use the DataFlex Flex-Keys to operate the screen.

This screen displays all of the data in the CUSTOMER file. The first field <CUSTOMER NUMBER> is actually the record number assigned automatically by DataFlex. Note that the use of record number this way is optional. User assigned CUSTOMER numbers or ID's could also be configured with only a small modification to the system. If an existing CUSTOMER number is entered into this field, the data from the CUSTOMER file for that record will display when the <RETURN> key is pressed:

EXAMPLE:

Enter a <2> and <RETURN> in window one, and the data for CUSTOMER two will display.

The CUSTOMER file is also INDEXED by the COMPANY NAME data field. This allows us to FIND customer records by placing the cursor either over the COMPANY NAME window, entering all or part of a company name and pressing FIND, or using the above procedure on the <CUSTOMER NUMBER> window.

EXAMPLE:

1) Clear the data windows with the CLEAR Flex-Key
2) Hit <RETURN> to move the cursor to the COMPANY NAME window
3) Type in "DATA" and execute a <FIND> command
4) The CUSTOMER data for "DATA ACCESS" will display.

Up to four fields on this screen could be indexed in this way on 8 bit systems (9 fields on 16 bit systems). Screens which have multiple files displayed can have up to this many indexes per file.

To ENTER new customers into the CUSTOMER database with this screen:

1) Clear the screen.
2) Strike <RETURN> to bypass the CUSTOMER number window. The number will be assigned automatically.
3) Type data into each window and press <RETURN>.
4) When all the required windows are filled, the record will be saved in the database

After reading the OPERATOR'S GUIDE, experiment with this screen until you feel comfortable with what the program does as you use the various keyboard commands. You should be able to:

1) FIND ANY RECORD BY EITHER KEY.
2) CREATE NEW RECORDS.
3) EDIT EXISTING RECORDS.
4) DELETE RECORDS.

CUSTOMER FILE DESCRIPTION:

The specifications (size, content and indexing) of the CUSTOMER data base file can be displayed or printed out with the FILEDEF program. The display/printout looks like this:

```
-------------------------------------------------------------------

        FILE DEFINITION LISTING FOR FILE #25
***************************************************
        FILE ROOT NAME     = CUSTOMER
        USER DISPLAY NAME = CUSTOMER FILE
        SHORT NAME         = CUSTOMER
***************************************************
        RECORD LENGTH = 128  (USED = 103)
        MAX NUMBER OF RECORDS = 100  (USED = 4)
        DELETED SPACE IS REUSED
        MULTI-USER RE-READ INACTIVE
***************************************************
FIELD   FIELD  FIELD    FIELD   DEC  MAIN  RELATES--TO
NMBR    OFFSET LEN      TYPE    PTS  INDEX FILE  FIELD
-----   ------ -----    -----   ---  ----- ----  -----
   1       1     30      ASCII   0     0     0     0       CUSTOMER
   2      31     30      ASCII   0     0     0     0       ADDRESS
   3      61     14      ASCII   0     0     0     0       CITY
   4      75      2      ASCII   0     0     0     0       STATE
   5      77     10      ASCII   0     0     0     0       ZIP
   6      87      1      NUMERIC 0     0     0     0       DISCOUNT
   7      88      4      NUMERIC 2     0     0     0       PUR MONTH
   8      92      4      NUMERIC 2     0     0     0       PUR YEAR
   9      96      4      NUMERIC 2     0     0     0       PROFIT
  10     100      4      NUMERIC 2     0     0     0       DUE

INDEX 1:  FIELD SEGMENTS:  <1>     <0>

-------------------------------------------------------------------
```

This display tells us the following things about the CUSTOMER file:

The CUSTOMER file is DataFlex file number 25

The ROOT FILE NAME (the one that will show on a disk directory) is "CUSTOMER"

When a file list is displayed to the operator it will show as "CUSTOMER FILE"

Each CUSTOMER record will take up 128 bytes in the data file (record length) and that of the 128, 103 bytes have been used.

The file can contain up to 100 customers (provided there is sufficient disk storage) but that only four records currently exist.

If a customer is deleted, the next customer to be added will be assigned the deleted customer's disk and record number automatically by DataFlex (deleted space is REUSED).

The CUSTOMER file is not expected to be used in a multiuser environment. (Multi-user Re-read is set INACTIVE).

There are 10 data fields in each customer record. Their lengths, data types, and names are also shown.

One index is active for the CUSTOMER file, composed of field <1> (the customer name), and field <0> (the system record number).

Note that the 4-byte NUMERIC fields will each hold 8 digits, 2 after the decimal point.

Notice that in this file the record number is used as a system assigned account number. This provides for maximum operator and system speed when finding a customer record (particularly since the operator would remember frequently used customer numbers). Where this doesn't work well, you could create an additional indexed field for user-assigned account ID's or not use any codes at all by always using the CUSTOMER NAME. The operator can easily FIND the record by CUSTOMER NAME since that field is also indexed. Although access by record number is fastest, the INDEXING in DataFlex is so fast that indexing should not be avoided for reasons of performance.

The record number <0> is made part of the index with CUSTOMER NAME to create unique index entries to allow two customers with same name.

THE <CUSTOMER> CONFIGURATION FILE

The file name of the CUSTOMER file configuration on your demo disk is "CUSTOMER.FRM". The configuration file looks like this:

```
/FORM
DATA ACCESS CORPORATION                                   CUSTOMER MASTER FILE
==============================================================================

        CUSTOMER NUMBER: <___.>

        NAME: <______________________________>

     ADDRESS:  ______________________________

        CITY:  ______________  ST: __  ZIP: __________

        DISCOUNT: _.%

        PURCHASES/MONTH $______.__  PURCHASES/YEAR $______.__

        PROFIT/YEAR $______.__

Windows with <          > can be used to FIND customers
/*
// This is a simple ENTER configuration for customer file maintenance.
// Since the windows are processed sequentially, the AUTOPAGE function can
// be used and the windows do not have to be explicitly named.
        OPEN CUSTOMER                       // All files must be opened
        ENTER CUSTOMER
          AUTOPAGE FORM
// Each ENTRY command is followed by the database element it references.
// The window is assigned by autopage.      After the command the
// options surrounded by {}.
          ENTRY CUSTOMER.RECNUM             {AUTOFIND,NOPUT}
// Translation:  Do data entry on the record number of the customer
// file.  Automatically FIND by this field when data in input, but
// do not put data back into the record buffer since record number
// can't be changed.
          ENTRY CUSTOMER.CUSTOMER
// Translation:  Do data entry on the customer name of the customer file
          ENTRY CUSTOMER.ADDRESS
          ENTRY CUSTOMER.CITY
          ENTRY CUSTOMER.STATE
          ENTRY CUSTOMER.ZIP
          ENTRY CUSTOMER.DISCOUNT
          ENTRY CUSTOMER.PUR_MONTH                    {DISPLAYONLY}
// Translation:  When the customer file is found, display purchases
// for the month in this window
          ENTRY CUSTOMER.PUR_YEAR           {DISPLAYONLY}
          ENTRY CUSTOMER.PROFIT             {DISPLAYONLY}
          RETURN        // return from entry section
        ENTEREND        // end of enter configuration
        ABORT           // abort after escape key.
        KEYPROC KEY.USER// Define action of user defined key
          OUTPUT FORM   // Output the page
          ENTAGAIN      // Return to same window
        RETURN          // End definition of user defined key
```

The HELP screens in the actual configuration have been eliminated in the above listing to save space, as have certain spacing lines in the screen image.

The top of the configuration file is the IMAGE format of the screen. DataFlex will display the form exactly as it appears in the first part of the configuration. First the page is named with the "/FORM", designating "FORM" as the name of the page. The DATA WINDOWS are numbered from left to right, top to bottom. Notice that the last four windows are numeric and that the last three contain exactly two places to the right of the decimal point. There are ten windows in this image format.

In this image, there is a one-to-one correspondence between the fields in the data file and the display windows on the CRT screen. Images do not need to display all fields from a file; a single data field from a file might be all that was needed to be displayed.

The line following the end of the screen contains a "/*". This mark indicates the end of the image format. It is a required element of the configuraton. It tells the DataFlex compiler where the end of the image is, and where the command part of the configuration begins.

The first command line opens the CUSTOMER file. All files that are to be used in a configuration must be OPENed. OPENs should be at the begining of the configuration.

The next line in the configuration "ENTER CUSTOMER" begins the ENTER Command Group. The ENTER command is a DataFlex macro that is really more like a complete program than just a command since it controls the whole operation of this configuration.

The next line, "AUTOPAGE FORM", tells the DataFlex compiler that we will be using the page named "FORM". By using AUTOPAGE, we don't have to specify each window on the page. They are assigned automatically in sequence by the command.

The next 10 lines are ENTRY commands. ENTRY commands work with the ENTER command and specify each data element and window. The window numbers can be omitted because of the use of the AUTOPAGE Command.

See the section of this manual on the ENTER Command Macro for a complete discussion of the functions and uses of ENTRY.

The first ENTRY command line uses two options:

AUTOFIND:	Executes a FIND using data in the window when the <RETURN> key is pressed.
NOPUT:	Do not put data BACK INTO the record buffer since the record number (the window data in this case) can not be changed.

Refer to the ENTER Command Macro section for a full list of options.

At this time, review each of the command lines to see how the file and field element names correspond to the windows in the screen image and the data file definition. Note that in line one, record number is connected to window one (account number). This is typical in files where the record number is used as a system-assigned account number.

The last line of the ENTER configuration for CUSTOMER FILE is an ABORT. When a file maintenance session is complete, pressing the ESCAPE key will cause the configuration to abort.

CUSTOMER FILE REPORT EXAMPLE:

You need a simple report to list our customers, their discount and total purchases. The following is the output of the REPORT program:

```
---------------------------------------------------------------------

                      CUSTOMER FILE

CUSTOMER                                    DISCOUNT  PUR/YEAR

    4  DATA ACCESS CORPORATION                 15%       100.00
    3  DYSAN CORPORATION                        5%     49,713.50
    2  MAXELL                                  20%     12,056.00
    1  MT MICROSYSTEMS, INC.                   30%     57,000.00
                                                      ----------
                                                      118,869.50

---------------------------------------------------------------------
```

This report is produced with the following report configuration:

```
/HEADER
                         CUSTOMER FILE
CUSTOMER                                           DISCOUNT      PUR/YEAR

/BODY RESIDENT
 _____. ______________________________          __.%    ___,___.__
/TOTAL
                                                                ----------
                                                             __,___,___.__
/*
OUTFILE                                //direct the output
OPEN CUSTOMER INDEX.1                  //open the customer file

REPORT CUSTOMER BY INDEX.1
//Print the customers in customer name order.

SECTION HEADER
  OUTPUT HEADER                        //simply output the page headings

SECTION BODY
  PRINT CUSTOMER.RECNUM                //customer id number is system assigned
  PRINT CUSTOMER.CUSTOMER              //customer name
  PRINT CUSTOMER.DISCOUNT
  PRINT CUSTOMER.PUR_YEAR
  OUTPUT BODY

SECTION TOTAL
  SUBTOTAL BODY.4                      //subtotal will take the accumulator
                                       //for any numeric window and print
                                       //it in another window.
                                       //print the total purchases for all
  OUTPUT TOTAL                         //customers in the database
REPORTEND
ABORT
```

THE VENDOR FILE:

The vendor file, like the CUSTOMER file, is a single file configuration. We will relate other files to it at a later time. The first step is to run the VENDOR configuration from the DEMO menu. Then examine the file definition for VENDOR and compare it with that of the CUSTOMER file.

```
        FILE DEFINITION LISTING FOR FILE #21
***************************************************
        FILE ROOT NAME    = VENDOR
        USER DISPLAY NAME = VENDOR
        SHORT NAME        = VENDOR
***************************************************
        RECORD LENGTH = 128  (USED = 112)
        MAX NUMBER OF RECORDS = 400  (USED = 15)
        DELETED SPACE IS REUSED
        MULTI-USER RE-READ INACTIVE
***************************************************
FIELD   FIELD  FIELD     FIELD   DEC  MAIN  RELATES--TO
NMBR    OFFSET LEN       TYPE    PTS  INDEX FILE  FIELD
-----   ------ -----     -----   ---  ----- ----  -----
   1       1    30       ASCII          1     0     0    VENDOR
   2      31    30       ASCII          0     0     0    ADDRESS
   3      61    14       ASCII          0     0     0    CITY
   4      75     2       ASCII          0     0     0    STATE
   5      77     9       ASCII          0     0     0    ZIP
   6      86     3       ASCII          0     0     0    PHONE-AC
   7      89     3       ASCII          0     0     0    PHONE_EXCHG
   8      92     4       ASCII          0     0     0    PHONE_NUMBER
   9      96     2       NUMERIC 0      0     0     0    TERMS
  10      98     4       NUMERIC 2      0     0     0    CREDIT_LIMIT
  11     102     4       NUMERIC 2      0     0     0    CREDIT_USED
  12     106     4       NUMERIC 2      0     0     0    PURCHASES
  13     110     3       DATE           0     0     0    LAST_PURCHASE

INDEX 1:  FIELD SEGMENTS:  <1>     <0>
```

Note that the use of record number as an account number is used here as well. The indexing is also very much the same as in the CUSTOMER data base file. The only real difference is the content of the fields. This is an important DataFlex concept: that the way information is dealt with remains constant even though the application may change.

The VENDOR file introduces us to the DATE data type and field format. Dates in DataFlex are kept in JULIAN format, or, the number of days since day zero. Since dates are kept as numbers we can use them in arithmetic expressions (add or subtract them). This is quite useful for ageing reports and due date calculations, etc. Even though dates are stored as numbers, they are always displayed in a DATE format window (__/__/__) automatically converted to the conventional MM/DD/YY format.

File maintenance on the VENDOR file uses the following configuration:

```
        ==========================================================
==================================================================

/FORM
DATA ACCESS CORPORATION                                 VENDOR MASTER FILE
======================================================================
        VENDOR NUMBER: <__.>

      COMPANY: <______________________________>
      ADDRESS:              ______________________________
         CITY:              ______________ ST: __  ZIP: _________

        PHONE: (___) ___-____

        TERMS (DAYS): __.    CREDIT LIMIT: $______.__
                        OUTSTANDING CREDIT: $______.__
                                            ------------
                          REMAINING CREDIT: $______.__
     TOTAL PURCHASES: $______.__
       LAST PURCHASE: __/__/__
Data in <  > can be used to FIND vendors
/*
        OPEN VENDOR                 // all files must be opened.
        ENTER VENDOR                // initalize enter macro.
          AUTOPAGE FORM             // automatically assign windows
          ENTRY VENDOR.RECNUM       {NOPUT,AUTOFIND}
// Translation, Do data entry with the record number of the vendor file
// automatically find record, do not update the record buffer since
// record number can't be changed.
          ENTRY VENDOR.VENDOR
// Translation, Do standard data entry on the vendor name of the vendor file.
          ENTRY VENDOR.ADDRESS
          ENTRY VENDOR.CITY
          ENTRY VENDOR.STATE
          ENTRY VENDOR.ZIP
          ENTRY VENDOR.PHONE_AC
          ENTRY VENDOR.PHONE_EXCHG
          ENTRY VENDOR.PHONE_NUMBER
          ENTRY VENDOR.TERMS
          ENTRY VENDOR.CREDIT_LIMIT
          ENTRY VENDOR.CREDIT_USED          {DISPLAYONLY}
// Translation, When the vendor file is FOUND, display credit used in this
// window.
          ENTRY (FORM.11-FORM.12)           {DISPLAYONLY}
// Translation, When a record is found, display the result of the calculation
// of credit limit (FORM.11) minus credit used (FORM.12) in this window.
          ENTRY VENDOR.PURCHASES            {DISPLAYONLY}
          ENTRY VENDOR.LAST_PURCHASE        {DISPLAYONLY}
          RETURN                    // return from entry section
        ENTEREND                    // end of enter command
        ABORT                       // stop after escape key.
// note: the entry command can only be used inside an enter or entergroup
KEYPROC KEY.USER                    // Define action of user defined key
        OUTPUT FORM                 // Output the page
        ENTAGAIN                    // Return to same window
        RETURN
```

Notice the similarity between this file and the one for CUSTOMERs. Each ENTRY command line links a data base field with a WINDOW.

Close to the bottom, we see something new: ENTRY (FORM.11-FORM.12). This is a calculated field that translates to:

> WHEN DISPLAYING FILE 21, SUBTRACT THE VALUE OF WINDOW 12 (FORM.12) FROM THE VALUE OF WINDOW 11 (FORM.11) AND PUT THE RESULT IN WINDOW 13.

There is not a report for the vendor file supplied with the sample configurations; this one is up to you. Create a report format for VENDOR and use the CUSTOMER report as a model. After making the report, add it to the MENU using the MENUDEF program.

After making your report, try your hand at a simple file definition and configuration using the AUTODEF Program. AUTODEF takes a screen IMAGE format created with a text editor and makes both the file definition and compilable configuration file from it. After making these, look at the file definition with the FILEDEF program and type out the configuration with the operating system TYPE command. Then compile the configuration.

THE INVENTORY FILE:
With the inventory file we start to see the true power of DataFlex. Each inventory part is supplied by a VENDOR, so the INVENTORY file is said to RELATE to the VENDOR file. When an inventory part is found (called into memory), the RELATED VENDOR record automatically comes with it. This relationship is automatically maintained by DataFlex.

```
DATA ACCESS CORPORATION                          INVENTORY MASTER FILE
======================================================================

      VENDOR NUMBER: __.      NAME: ______________________________

     VENDOR PART ID: _______________

       YOUR PART ID: _______________

        DESCRIPTION: ___________________________________

     STOCK QUANTITY:   _____.             MINIMUM QUANTITY: ___.

   REORDER QUANTITY:   _____.               QTY ON ORDER: _____.

       SALES/MONTH: $________.__  SALES/YEAR: $________.__

       PROFIT/YEAR: $  ______.__

        LIST PRICE: $  ______.__     WHOLESALE PRICE: $______.__

      AVERAGE COST: $  ______.__ STOCK VALUE: $________.__

======================================================================
```

When running the inventory data entry screen, the first two fields are actually out of the VENDOR file. You can find the vendor of interest on this screen by either key, just like you could in the VENDOR screen. You can then move the cursor down to VENDOR PART ID: to find the first (or any) part for that vendor. The vendor number and then vendor's part number make up a "two segment key", that is, you must find the vendor, then, put in all or part of the vendors part number to find a record by that index. To find the first part for any vendor, enter the VENDOR NUMBER or NAME and hit the "SUPERFIND" key.

The INVENTORY file is also indexed (able to find records by) "YOUR PART ID" and "DESCRIPTION". You can put the cursor in any of these windows and find a particular part!

When you are creating a new part, find the vendor first, then fill in the other windows like you did in the other screens. This configuration will also allow you to create new vendors from this inventory screen. Just enter a new vendor name with no vendor number, and when you SAVE the part, a new vendor will be created. Later we will see how to disable this feature if you want.

INVENTORY FILE DEFINITION:

```
        FILE DEFINITION LISTING FOR FILE #22
****************************************************
        FILE ROOT NAME     = INVT
        USER DISPLAY NAME = INVENTORY FILE
        SHORT NAME         = INVT
****************************************************
        RECORD LENGTH = 128  (USED = 104)
        MAX NUMBER OF RECORDS = 100  (USED = 33)
        DELETED SPACE IS REUSED
        MULTI-USER RE-READ ACTIVE
****************************************************
FIELD   FIELD   FIELD   FIELD   DEC  MAIN   RELATES--TO
NMBR    OFFSET  LEN     TYPE    PTS  INDEX  FILE  FIELD
-----   ------  -----   -----   ---  -----  ----  -----
  1       1      2      NUMERIC 0     1      21     0     VENDOR
  2       3     15      ASCII         1       0     0     VENDOR PART_ID
  3      18     15      ASCII         2       0     0     PART_ID
  4      33     35      ASCII         3       0     0     DESCRIPTION
  5      68      3      NUMERIC 0     0       0     0     QUANTITY
  6      71      2      NUMERIC 0     0       0     0     MINIMUM
  7      73      3      NUMERIC 0     0       0     0     REORDER
  8      76      3      NUMERIC 0     0       0     0     ON_ORDER
  9      79      5      NUMERIC 2     0       0     0     SALES_MONTH
 10      84      5      NUMERIC 2     0       0     0     SALES_YEAR
 11      89      4      NUMERIC 2     0       0     0     PROFIT_YEAR
 12      93      4      NUMERIC 2     0       0     0     LIST_PRICE
 13      97      4      NUMERIC 2     0       0     0     WHOLESALE_PRICE
 14     101      4      NUMERIC 2     0       0     0     AVG_COST

INDEX 1:  FIELD SEGMENTS:  <1>     <2>
INDEX 2:  FIELD SEGMENTS:. <3>
INDEX 3:  FIELD SEGMENTS:  <4>     <0>
```

The first field "VENDOR NUMBER" **RELATES** to the record number (field zero) of the vendor file (#21). This is all that's required to RELATE the inventory file to the vendor file and it essentially makes the vendor file a logical extension of the inventory file.

The INVENTORY file has THREE indexes. The first index is by vendor number (field one) AND vendor part id (field two). This combining of fields to make one index is called a multi-segment index. Since record number is NOT appended to this index, two parts can't have the same vendor AND part number, which would be a duplicate record.

Index TWO is on field 3, or YOUR PART ID, allowing the same part to be known by two different part numbers, yours and the vendor's. Could two parts have the same part number? If not, how could you allow that if you wanted? Answer: No, two parts could not have the same number, but if they could, the circumstance could still be dealt with effectively by appending field ZERO (the record number) to the index, rendering the composite index entry unique despite the duplication.

Index THREE allows access by DESCRIPTION. Notice the "MAIN INDEX" column of the file definition. The main index is the number of the index that will be used when you FIND by that field. In this file there are no fields that are used in more than one index, but where there are, the main index becomes important.

THE INVENTORY ENTER CONFIGURATION: (some lines are deleted from the image area to conserve space)

```
/FORM
DATA ACCESS CORPORATION                                   INVENTORY MASTER FILE
=======================================================================

       VENDOR NUMBER:<__.>     NAME:<______________________________>

      VENDOR PART ID:<_______________>

        YOUR PART ID:<_______________>

         DESCRIPTION:<___________________________________>

      STOCK QUANTITY:   _____.            MINIMUM QUANTITY: _.

    REORDER QUANTITY:   _____.              QTY ON ORDER: _____

        SALES/MONTH: $________.__  SALES/YEAR: $________.__

        PROFIT/YEAR: $  _______.__

         LIST PRICE: $  _______.__    WHOLESALE PRICE: $______.__

       AVERAGE COST: $  ______.__ STOCK VALUE: $________.__

/*

// This enter configuration is a little more sophisticated since it deals
// with two files.  The inventory file RELATES to the vendor file.
        OPEN INVT            // All files must be OPENED before use.
        OPEN VENDOR          // The order of opening the file is
                             // not important.

        ENTER INVT  VENDOR   // ENTER will maintain both INVT and VENDOR.
                             // Since INVT relates TO VENDOR it MUST be
                             // first on the ENTER line.
                             // INVT is the main file.

        AUTOPAGE FORM        // Set up to automatically assign windows.
 // These fields use the AUTOPAGE option--the windows are automatically
 // assigned to the FORM page.
         ENTRY VENDOR.RECNUM           {AUTOFIND}
// Translation, do data entry on the record number of the vendor file.
// Automatically try to find the vendor by number.
// Note that the element of the related TO file is used here
// and automatically PUT to the VENDOR field of the INVT file.
// The related TO element should always be used in a case like this.
```

(This configuration is continued on the next page.)

```
         ENTRY VENDOR.VENDOR           {SKIPFOUND,FINDREQ}
// Translation, Do data entry on the vendor name of the vendor file
// Skip this entry if a record has already been found and REQUIRE
// that a vendor be found to continue.
         ENTRY INVT.VENDOR_PART_ID    {SKIPFOUND,RETAIN}
// Translation, Do data entry on the vendor part id of the inventory
// file.  Skip this if a record has been found (preventing changes)
// and retain the entry after a clear window command.
        ENTRY INVT.PART_ID {RETAINALL,REQUIRED}
        ENTRY INVT.DESCRIPTION {CAPSLOCK}
        ENTRY INVT.QUANTITY
        ENTRY INVT.MINIMUM
        ENTRY INVT.REORDER
        ENTRY INVT.ON ORDER
 // These fields do not use autopage--they name each window.
 // Either form is acceptable.
        ENTRY INVT.SALES MONTH FORM.10 {DISPLAYONLY}
        ENTRY INVT.SALES YEAR FORM.11 {DISPLAYONLY}
        ENTRY INVT.PROFIT_YEAR FORM.12
        ENTRY INVT.LIST_PRICE FORM.13
        ENTRY INVT.WHOLESALE_PRICE FORM.14
        ENTRY INVT.AVG_COST FORM.15
        ENTRY (INVT.QUANTITY*INVT.LIST_PRICE) FORM.16
// The line above displays a calculated field.
        RETURN // return is required at the end of the entry section.
        ENTEREND       // as is ENTEREND.
        ABORT          // After the ESCAPE key is hit, abort.

KEYPROC KEY.USER       // Define action of user defined key
        OUTPUT FORM    // Output form to list
        ENTAGAIN       // Return to same window
        RETURN
```

Notice that there are now two files OPENED on the first two lines of the configuration section, INVT and VENDOR. This tells ENTER to use both the VENDOR and INVENTORY files in this screen. The ORDER in which these files are listed on the ENTER line is important. The files which are RELATED TO by the others go to the right of the files which relate TO them. The first file is called the MAIN file. The MAIN file is the one that relates (directly or indirectly) to all the others. INVT is the main file in this configuration. (there can be files that aren't related to at all).

The first two fields are from the VENDOR file while all the others are from the inventory file. When the same (related) field is present in both files, such as; RECORD NUMBER in the VENDOR file and VENDOR NUMBER in the INVENTORY file, ALWAYS use the field that is related TO by the other(s) in the ENTER configuration. That's why VENDOR.RECNUM is used in the first window instead of INVT.VENDOR. VENDOR NUMBER in the INVENTORY file will be automatically updated when a record is saved.

This ends the sample section of the demo for your initial familiarization. The rest of the demo should be studied after you master basic operations with some application work of your own.

APPLICATION STRUCTURE:

So far, we have been considering only one or two files at a time, but to really understand what is happening in our order entry application. Let's look at the whole picture:

Select "INVOICE ENTRY" from the demo menu. The invoice screen will look like this:

```
DATA ACCESS CORPORATION                                           INVOICE
=========================================================================

INVOICE NUMBER: _____.                          CUSTOMER# ___.

        BILL TO: ______________________________
                 ______________________________
                 ______________  __  __________

DISC  TERMS   DATE       VIA           SLSMN   CUSTOMER ORDER
 _.    __.   __/__/__    ____________  _____   ______________
.........................................................................
   PART ID       DESCRIPTION                       PRICE   QTY   AMOUNT
   ------------- ----------------------------- -------- ---- -------
1. _____________ _____________________________ _____.__ ___. _____.__
2. _____________ _____________________________ _____.__ ___. _____.__
3. _____________ _____________________________ _____.__ ___. _____.__
4. _____________ _____________________________ _____.__ ___. _____.__
5. _____________ _____________________________ _____.__ ___. _____.__
6. _____________ _____________________________ _____.__ ___. _____.__
                                                            ==========
                                                  TOTAL $_________.__
-------------------------------------------------------------------------
```

To create a new invoice, enter the invoice number. You may then ENTER the CUSTOMER NUMBER in window two or the CUSTOMER NAME in window three and do a FIND. Once you have found a customer, the CUSTOMER data will display. This will become the default for the ship-to address; you may change these fields if you wish. The cursor will then move to invoice header area and take the default values (these are coming from the CUSTOMER file). In the LINE ITEM section the object is to find an inventory item by either of the two keys, and then ENTER the QTY sold. A default for the unit price will come from the inventory file less the discount in the CUSTOMER file. The other affected windows will be updated and you can save the line item. You may use the arrow keys to move up and down through the items. The delete record key deletes an entered line item. Press the ESCAPE key to print the invoice.

There are five files open in the invoice screen; Customer, Vendor, Inventory, Invoice header, Line items (inventory transactions). The structure of the files looks like this:

```
      VENDOR      CUSTOMER
         |            |
         |            |
    INVENTORY         |
         |            |
         |            |
       LINE ITEMS-----'
```

The configuration that produces a multi-line multi-file report like this is advanced "Flex-gramming". It is included on your disks as the file "INVOICE.FRM". You should understand basic operations with ENTER and REPORT before trying a configuration like INVOICE, which uses command described in the next section.

The files, screens and reports that make up the demo are:

FILE NUMBER	FILE NAME	REPORT CONFIGURATION	ENTER CONFIGURATION
1	SYSFILE (SYSTEM FILE)	NONE	DATE.FRM
21	VENDOR	VENRPT.RPT	VENDOR.FRM
22	INVT (INVENTORY)	INVT.RPT	INVT.FRM
25	CUSTOMER	CUSTRPT.RPT	CUSTOMER.FRM
29	INTRAN (LINE ITEMS)	MOVEMENT.RPT	INVOICE.FRM

THIS PAGE INTENTIONALLY LEFT BLANK

DIRECT CONTROL OF DATA ENTRY

The DataFlex Forms command group controls screen image PAGEs, including display of the PAGE image, filling the windows with data, moving the data from windows to the record buffer, formatting display of data, and clearing both data and images from the screen.

PAGE Command

The action of the PAGE command is to display any labeled PAGE of a configuration on the screen. PAGEs are labeled through the use of a "pagename" as discussed in the chapter on "Elements of Configurations". The prerequisite for the PAGE command is that at least one defined and labeled image page be present as part of the configuration being processed. See the section on "Formatting With Images" for details of the page configuration.

FORMAT:

```
        PAGE pagename
```

EXAMPLES:

```
        PAGE firstscreen
        PAGE secondscreen
        PAGE pageheading
        PAGE subheading
        PAGE reportbody
```

Any MOVE or DISPLAY to a page will be immediately displayed in the appropriate window when a PAGE command is executed for that page. It is important to note that since windows are variables and can be used just like any other defined and typed variable in DataFlex, data can be MOVEd to or from a window without the necessity of displaying the PAGE image.

ACCEPT Command

ACCEPT places the cursor in an image "window" and waits for keyboard input. Further, the <u>only</u> valid argument for the ACCEPT command is a window, referenced either as the format PAGENAME.# where "#" is the number of the window in the page, or as a named window with the window name established by the NAME definition command.

On execution, ACCEPT displays the operator's keyboard input into a window (as specified above). It can, optionally, pass the input data to any legal argument in the DataFlex configuration using the optional "TO argument" extension as part of the execution of the same command. The DataFlex field edit keys are active during an ACCEPT command.

When using ACCEPT, remember that the window reference itself is a valid variable that can be used throughout a configuration. Data does not need to be moved to another variable for processing in calculations, data base operations etc. Useful amounts of memory can be conserved by using the window as a variable to contain data; if a variable is defined only to hold the data input from a window, a dual overhead is created to store the same information.

FORMAT:
```
        ACCEPT window {TO argument}
```

EXAMPLES:
```
        ACCEPT screen.3

        ACCEPT screen.3 TO inventory.description

        ACCEPT window7 TO inventory.qty  (window7 previously
                                          defined by NAME command

        ACCEPT screen.3 TO screen.10
```

Where the optional {TO argument} is omitted from the ACCEPT command line, the data in "window" can be processed in any normal manner. Where the {TO argument} is used, an automatic MOVE command is executed when the input is received, placing the data from "window" into "argument". The "ACCEPT window TO argument" election itself has no effect on the contents of the window, which will hold the same data as the argument after the command is executed. The window argument may be left out of the ACCEPT command line by using the AUTOPAGE command. (see AUTOPAGE)

If data already exists in a window from a previous ACCEPT, MOVE or DISPLAY, it will be the default entry for the next ACCEPT. If the appropriate PAGE is not displayed at the time that an ACCEPT is executed, the PAGE will be displayed by the ACCEPT.

DISPLAY Command

DISPLAY explicitly outputs data to a window in the same manner as ACCEPT allows input. Accordingly, the <u>only</u> valid output destination for DISPLAY is a window in an image addressed either as PAGENAME.# where "#" is the sequential number of the window in a page image, or as a name established with the NAME command. DISPLAYed data will not clear when the CLEAR ALL Flex-Key is pressed; only data displayed by the ENTRY command will respond to the CLEAR ALL Flex-Key.

FORMAT:
```
        DISPLAY argument TO window
```

EXAMPLES:
```
        DISPLAY invt.description TO screen.3

        DISPLAY member.name TO mem_name
        Where "mem name" was previously created with the NAME command
```

The window reference may be left out of the DISPLAY command by using AUTOPAGE. (see AUTOPAGE)

AUTOPAGE Command

AUTOPAGE performs an automatic sequential numbering of windows on a PAGE for the ACCEPT, DISPLAY and PRINT commands. ACCEPT and DISPLAY commands which follow an AUTOPAGE command will reference windows in sequence from left to right, top to bottom. The argument for the AUTOPAGE command is any pagename valid for the current configuration. If AUTOPAGE is only to be used on a portion of a PAGE, the number of the starting window may optionally be added to the command. The {} braces shown below should not be typed--only the number of the starting window. It is not necessary to process every window in the PAGE after an AUTOPAGE command, but the list of windows to process must be continuous (no gaps) after each AUTOPAGE command.

AUTOPAGE allows the use of the ACCEPT, DISPLAY and PRINT commands without the necessity of specifying the window where the command action is to take place.

FORMAT:

```
        AUTOPAGE pagename {#}
```

EXAMPLES:

```
        AUTOPAGE employee_record            //Display the screen
              DISPLAY emp.name              //Display the 1st window
              DISPLAY emp.address           //Display the 2nd
              DISPLAY emp.city              //Display the 3rd
              DISPLAY emp.phone             //Display the 4th
              ACCEPT TO data.verified       //Accept input to the 5th

        AUTOPAGE employee_tax_status 8      //Display the screen
              DISPLAY emp.marital           //Display the 8th window
              DISPLAY emp.exemptions        //Display the 9th window
```

CLEARFORM and BLANKFORM Commands

These commands remove data from image windows. They can address an entire image (by pagename), an individual window or a range of windows.

For example, after data has been entered to a screen image and saved to the disk, one of these commands can be executed to clear the input data from the windows and ready the image for the input of more data.

The commands differ as follows: CLEARFORM removes all data from the image and displays the "fill character" in the window as defined in the SETSCREE Utility; BLANKFORM removes all data in the image and displays ASCII space characters in the windows. The two different forms of operation are provided to allow as much flexibility as possible in application design. Most configurations will probably use CLEARFORM. The DataFlex READ configuration uses BLANKFORM when displaying the field tag names for reference.

FORMAT:

```
        CLEARFORM pagename          (clear entire page)
        BLANKFORM pagename

        CLEARFORM window            (clear a window)
        BLANKFORM window

        CLEARFORM window.n1 THRU window.n2
        BLANKFORM window.n1 THRU window.n2
                                    (clear a range of windows
                                    starting at window.n1 and
                                    stopping at window.n2)
```

OUTFILE Command

OUTFILE is used to identify the sequential device (printer, screen, a disk file) which is to receive data controlled by the OUTPUT command. The argument for the OUTFILE command is any valid file or device name addressable by the operating system. If no OUTFILE command exists in a configuration, and the OUTPUT command is used, output will be directed to the printer port. If the OUTFILE command is provided, but with no argument, this means that an argument provided with the operating or menu command which runs the configuration will designate the device (the operating system command FLEX DAILYREPORT CON: would cause the configuration DAILYREPORT to be output to the screen). If no argument is entered on the command line, output will again go to LST: (the printer in most systems). Finally, the OUTFILE command may itself have an argument in the configuration as shown below. Quotation marks are required for literal arguments.

FORMAT:

```
        OUTFILE {string_arg}
```

Where "string_arg" is a literal text string in quotation marks or a string variable containing the name of a legal output device or file.

EXAMPLES:

```
        OUTFILE "LST:"      (output to printer (list device))

        OUTFILE "abc.dat"   (output to file ABC.DAT)

        OUTFILE destdev     (output to device or file  indicated by
                            the contents of string variable DESTDEV

        OUTFILE             (looks for argument from command line)
```

The OUTFILE command permits output in the same configuration with other operations. For example, after data for an invoice has been input, it can be sent to a printer for demand printing.

OUTCLOSE Command

After a file or device is established with OUTFILE, OUTCLOSE is used to "close" the file/device when the configuration is being terminated.

This command uses no argument, since it operates on any device or file which is open.

FORMAT:

```
        OUTCLOSE
```

OUTPUT Command

OUTPUT addresses a "pagename" argument and transfers the entire page referenced by "pagename" to OUTFILE. All data in the page--formatting, prompts, and data, are sent to OUTFILE. See OUTFILE command.

FORMAT:

```
        OUTPUT pagename_arg
```

EXAMPLE:

```
        OUTPUT invoice
```

NAME Command

NAME is used to identify windows in a page with a specific identifier rather than PAGENAME.#. There are two forms of the command. A window can be explicitly NAMEd:

FORMAT:

```
        NAME page_name.# window_name
```

Or a group of windows in a page can be NAMEd:

```
        NAME page_name window_name1 window_name2 ... window_name8
```

Using the second form of the command, multiple windows can be NAMEd on a single command line; multiple NAME command lines can be used to NAME all of the windows in a page, if the names of all the windows do not fit conveniently on a single line. Care should be taken to avoid using any command or option word of DataFlex (such as NAME, PAGE, date, number, etc.) as a window NAME. The names are applied to windows in order from top left to right, and then down by line.

EXAMPLES:

```
        NAME invt.6 quantity

        NAME invt part_id descr vndr cost price quantity
```

Use of an AUTOPAGE with a starting window # enables the NAMEing of a group of windows starting with that window number. In this instance, the pagename should not be repeated after the AUTOPAGE command. Again, multiple NAME lines may be used if necessary to NAME all the windows.

EXAMPLE:

```
        AUTOPAGE invt 7
        NAME quantity stock profit total
```

FORMAT Command

Each DataFlex window may have one or more format and data verification options. The options are applied by use of a FORMAT command specifying the window, the desired options, and any applicable parameters. NOTE: In this instance, the brackets {} used to delimit the options are literal, and are to be typed.

FORMAT:

```
        FORMAT window_arg {option1, option2 ... optionN}
```

"Window_arg" can be in the form "pagename.#" or a named reference established with the NAME Command. The following options are available for window formatting:

FUNCTION	SYNTAX
DECIMAL FORMAT	{POINTS=n} where "n" is the number of places to the right of the decimal point in the window. This can reformat a window or can be used to configure an integer-only numeric field by setting a value of zero (0) for "n".
UPPER CASE	{CAPSLOCK} This option accepts upper case entry and converts lower case entry to upper case.
RANGE CHECK	{RANGE = #1, #2} {RANGE = mm/dd/yy, mm/dd/yy} Provides an automatic check of entered data and declares an error and returns the cursor to the window if the date or numeric value entered does not fall within the values specified in the RANGE command, where the first value is the low end of the range, and the second is the high end. The upper and lower values in the command are included within the valid range.

VALIDITY CHECK	{CHECK="entry1¦entry2¦entryn"}{CHECK=string_var} Provides an automatic check of entered data and declares an error and returns the cursor to the window if the entry does not match any substring in the CHECK command. Where an eligible entry has fewer characters than the window, the excess characters **must** be specified as spaces (shown below as ƀ). The vertical bars (¦) shown below are recommended optional delimiters. Delimiters, if used, should be characters unlikely to be input erroneously by operators. The command may alternatively reference a previously-defined in-memory string variable (not a window or database element):
	{CHECK="YES¦NOƀ¦Yƀƀ¦Nƀƀ"}
	{CHECK=password}

Calculations and string manipulations on data in formatted windows should be avoided, since the formatting can change the apparent "value" of the contents for purposes of the calculation or manipulation. Calculations and manipulations should be done on the source data (database element, literal string, in-memory variable, etc.) from which the window was filled to begin with, with replacement of the results to the window, if desired for output purposes.

The following additional options are valid for formatting output for the DataFlex PRINT command only. Like the other format options, they are set using the FORMAT command.

BLANK SUPPRESS	{SUPPRESS} Strips spaces from output (leading spaces in numeric data and trailing spaces in alphanumeric).
CHARACTER FILL	{FILL="x"} Fills in spaces remaining after data is put in window with the character specified in "x".
SIGN ON RIGHT	{SIGNRT} Places a minus sign on the right side of numeric values less than 0.
FLOATING DOLLAR	{FLOAT$} Places a floating dollar sign ($) on the left of a numeric value.

Multiple formatting options on a command line are enclosed within a single pair of braces and separated by commas:

EXAMPLE:

```
        {CAPSLOCK,CHECK="YES|NO "}.
```

PRINT Command

The PRINT command moves data to a window and uses special formatting options where they are included in the configuration. In particular, the PRINT command is the only command which can update SUBTOTAL accumulators in the REPORT macro.

FORMAT:

```
        PRINT arg {TO window}
```

EXAMPLES:

```
        PRINT invt.descr
        PRINT invt.descr TO BODY.2
        PRINT payroll_amt TO BODY.3
```

Note in the first example above that an AUTOPAGE command must be in force, since the command contains no window reference.

ENTERGROUP

The ENTER macro provides a complete data entry environment. The ENTERGROUP command is intended for inclusion in more custom-programmed configurations in which, for one reason or another, it may have been elected not to use the ENTER macro. The first use of ENTERGROUP will load a set of predefined key procedures that define the operation of the function keys: NEXT RECORD, PREVIOUS RECORD, BACK WINDOW, CLEAR, and SAVE. SAVE, however, will not save the record under ENTERGROUP, as it will under ENTER. Rather, it will simply go to the end of the ENTERGROUP and resume execution from there. Likewise, CLEAR will clear only the screen under ENTERGROUP, rather than clearing both the screen and the record buffer, as it does under ENTER. SAVE and CLEAR can, of course, be redefined to provide whatever action is appropriate to the application, as can all the other Flex-Keys, using KEYPROC procedures (see the section on "Using Function Keys".

Multiple ENTERGROUPs can be included in the same configuration. Each group must be ended with an ENDGROUP command. The use of multiple key access to records is fully supported by ENTERGROUP.

FORMAT:

```
        ENTERGROUP
              ENTRY commands
        ENDGROUP
```

Any changes in the ENTRY windows will be moved to the record buffer at the end of the enter group unless NOPUT was used on the windows.

EXAMPLE:

```
        ENTERGROUP
              ENTRY INVT.ID                    {AUTOFIND}
              ENTRY INVT.DESCRIPTION           {FINDREQ,SKIPFOUND}
              ENTRY INVT.VENDOR                {DISPLAYONLY}
              ENTRY INVT.PRICE
        ENDGROUP
```

This would allow FINDS in the INVT file by either ID or DESCRIPTION and allow the PRICE to be changed. The record buffer would be edited with PRICE but the record would NOT be saved.

All ENTRY formatting options are available in an ENTERGROUP, as is the ENTDISPLAY command, described in the section on "Data Entry". The SUPERFIND and DELETE Flex-Keys, however, are not defined, as they are in ENTER, because the more flexible possible applications of ENTERGROUP require application-specific programming of these functions, as they do not in ENTER. These Flex-Keys can be programmed with KEYPROC commands, as described in the section of this manual on "Using Function Keys". If it is desired for some reason to alter the pre-programming of other Flex-Keys as provided in ENTERGROUP for a specific application, those other Flex-Keys also can be re-programmed with KEYPROC commands.

Below is a sample configuration using one ENTERGROUP. It is the same ENTER configuration as is shown in the "Programming Tutorial" section of this manual, except that that example used the ENTER macro, and the one on the next page uses an ENTERGROUP. The differences between the two versions of the configuration are instructive.

```
/SCREEN
............................ PERSONNEL FILE .........................


                      Last             First       MI
     EMPLOYEE NAME :  <_______________ __________ __>

          ADDRESS1 :   _________________________
          ADDRESS2 :   _________________________
              CITY :   _______________  ST : __   ZIP :  _________

       SOC. SEC. # :  <_________>

        DATE HIRED :   __/__/__

          PAY RATE :  $_____.__

          PAY TYPE :   _ (H=Hourly or S=Salaried)

Enter a Name or Soc. Sec. # and press FIND to display record,
OR press <ESC> to exit and return to the DataFlex Menu...
/*
OPEN PEOPLE
START:                                                   //begin loop
   ENTERGROUP                                            //entergroup
      AUTOPAGE SCREEN
      ENTRY PEOPLE.LNAME
      ENTRY PEOPLE.FNAME
      ENTRY PEOPLE.MI
      ENTRY PEOPLE.ADDR1
      ENTRY PEOPLE.ADDR2
      ENTRY PEOPLE.CITY
      ENTRY PEOPLE.ST
      ENTRY PEOPLE.ZIP
      ENTRY PEOPLE.SSAN
      ENTRY PEOPLE.DATE
      ENTRY PEOPLE.PAYRATE
      ENTRY PEOPLE.PAYTYPE {CAPSLOCK,CHECK="HS"}
   ENDGROUP                                              //endgroup
   SAVE PEOPLE                                           //save record
   CLEAR PEOPLE                                          //clear buffer
   CLEARFORM                                             //clear screen
GOTO START                                               //begin loop
KEYPROC KEY.ESCAPE                                       //define esc key
ABORT                                                    //to abort
```

THIS PAGE INTENTIONALLY LEFT BLANK

MANIPULATING STRINGS

The DataFlex String commands allow manipulation of various forms of text string data. They can be used on literal string arguments, windows, string variables, or data base elements. If a non-string variable is used in the argument for a string command, it will be converted to a string before the string operation. String variables are declared using the STRING command, which is described in the chapter on Elements of Configurations. That section should be reviewed and understood prior to attempting the use of commands described in this chapter.

In most cases, the string commands act upon an existing string, specified by its variable name, or entered as a literal, and move the result of the action to a destination variable. The word "TO" is the designator preceding the destination variable which is to receive the output of the command. Certain string commands return the results of their processes back to the source variable name. These "replacing" string commands should be used with due regard for the fact that the original contents of the source string will not be available after execution of the command.

The general format of DataFlex string commands is:

```
        COMMAND src_string_var TO dest_string_var length position
```

Not all commands require the length and position specifiers, which are integers. DataFlex string commands can use the same variable name as both source and destination for the command action.

EXAMPLE:

```
        LEFT a_string TO a_string 10
```

will take the leftmost ten characters of a_string and place the result back into a_string.

LEFT Command

The LEFT command extracts a specified number of characters from a string starting with the leftmost character and places the result of the extraction into a variable.

FORMAT:

LEFT source_string TO dest_string ##
Where LEFT is the command, source_string is the string argument being acted upon, TO designates dest_string, and ## is an integer number or variable which specifies the length of the sub-string to be extracted starting from the beginning of source_string. If ##<1, the command will output a null string (no characters). If ## is greater than the length of the source string, the command will output the entire source string.

EXAMPLES:

Where "old_string" has a value of "ABCDEFGHI"
LEFT old_string TO new_string 5
would return a value of "ABCDE" in new_string

LEFT string TO first_char 1
would return the first character of "string" in the variable "first_char"

RIGHT Command

The RIGHT command extracts a specified number of characters from a string starting with the rightmost character and places the result of the extraction into a variable.

FORMAT:

RIGHT source_string TO dest_string ##
Where RIGHT is the command, source_string is the string argument being acted upon, TO designates dest_string, and ## is an integer literal or variable which specifies the length of the sub-string to be extracted starting from the right end of source_string. If ##<1, the command will output a null string (no characters). If ## is greater than the length of the source string, the command will output the entire source string.

EXAMPLES:

Where "old_string" has a value of "ABCDEFGHI"
RIGHT old_string TO new_string 5
would return a value of "EFGHI" in new_string

RIGHT string TO last_char 1
would return the last character of "string" in the variable "last_char"

MID Command

The MID string command provides the same sub-string extraction capability as found in the LEFT command, but MID's extraction takes place from a specified point within the string, not just from the end. MID adds a starting position argument to the syntax used for LEFT and right.

FORMAT:

> **MID source_string TO destination_string ##1 ##2**
> Where MID is the command, source string is the string being acted upon, TO designates the destination_string variable, ##1 is the length of the sub-string to extract and ##2 is the starting position in source_string where the extraction is to begin. Both ##1 and ##2 may be literal integers or variables.

EXAMPLE:

> Where OLD_STRING has a value of "ABCDEFGHI"
> **MID old_string TO new_string 5 2**
> would return a value of "BCDEF" in NEW_STRING

If ##1 is less than 1, the command will output a null string (no characters). If ##1 is greater than the length of the source string, the command will output the entire source string to the right of, and including, position ##2, provided ##2 is >0 and less than or equal to the source string length. If ##2<1 while ##1 is greater than the length of the source string, the entire source string will be output, while if ##2 is greater than the length of the source string, the rightmost character only of the source string will be output.

ASCII Command

The ASCII command takes the ASCII value of the first character of a source string and places the result in a numeric variable.

FORMAT:

> **ASCII string TO numeric_variable**

EXAMPLE:

> Where INITIAL has a value of "Abernathy"
> **ASCII initial TO init_val**
> will return a value of 65 in "INIT_VAL".

CHARACTER Command

The CHARACTER command takes an integer (0 to 127) and converts it to its equivalent ASCII character, then places the result into a string variable. This is particularly useful for creating strings needed for printer or other device control where the string values cannot be conveniently created from the keyboard.

FORMAT:

```
        CHARACTER integer TO string_var
```

EXAMPLES:

```
        Where LEAD_IN has been declared as a string variable
        CHARACTER 27 TO lead_in
        will place a value of 27 (the ESC character) in LEAD_IN

        Where EIGHTLPI has been declared as a string variable
        CHARACTER 65 TO arg
        will place an "A" (ASCII code 65) in EIGHTLPI
```

POS Command

The POSition command scans a source string for the occurrence of a designated "target" substring, and returns an integer numeric result which is the starting position of the target string within the source string. In the event the target substring is not found within the source string, the POS command stores a zero in the specified variable.

FORMAT:

```
        POS string1 IN string2 TO num_var
```

Where POS is the command, STRING1 is the string to be scanned for, IN designates STRING2, STRING2 is the string to be scanned for an occurrence of STRING1, TO designates the destination numeric variable NUM_VAR which will contain the result.

EXAMPLE:

```
        Where STRING1 = "DEF", and STRING2 = "ABCDEFGHI"
        POS string1 IN string2 TO posit
        will return a value of 4 in POSIT.
```

LENGTH Command

LENGTH returns the length of a designated string as an integer number in a numeric variable. LENGTH counts the number of printable characters and spaces in the argument. LENGTH will give a different result for what may seem like the "same" data according to whether the argument is a database element, a window, or a variable. See the discussion at the end of this chapter on "String Operations".

FORMAT:

> **LENGTH string TO num_var**
> Where LENGTH is the command, STRING is the string whose length is to be determined, TO designates the numeric variable to contain the result of the command's action.

EXAMPLE:

> Where STRING="ABC"
> **LENGTH string TO how_long**
> will return a value of 3 to HOW_LONG

PAD Command

The PAD command is used to make strings of specified length. The first argument identifies the variable which is the source of the data to be PADded, and the second identifies the destination string in which the PADded result is stored after processing. The third argument is an integer value or variable designated as the length of the string to result from the execution of the PAD command. If the original string is shorter than length, spaces will be added to string to extend it to length. If the string is longer than length, characters in excess of length will be eliminated

FORMAT:

> **PAD source_string TO dest_string ##**
> Where PAD is the command, SOURCE_STRING contains the string to be PADded, DEST_STRING contains the PADded result of the command, and ## is the integer value or variable which specifies the number of characters (including spaces, if required) that will be contained in the resultant string.

EXAMPLE:

> Where ALPHA_STRING = "ABC", and spaces are represented as "b̸"
> **PAD alpha_string TO beta_string 5**
> will store "ABCb̸b̸" in variable BETA_STRING
>
> Where BEFORE = "ABCDEFG"
> **PAD before TO after 3**
> will store "ABC" in variable AFTER

TRIM Command

The TRIM command strips both leading and trailing blanks, if present, from the contents of a string variable. Embedded blanks are not affected.

FORMAT:

> **TRIM source_string TO dest_string**
> Where SOURCE_STRING contains the string to be operated on, and DEST_STRING will contain the TRIMmed string. SOURCE_STRING and DEST_STRING may be the same if replacement is desired.

EXAMPLE:

> Where spaces are represented as "b̸", and LONGSTRING = "b̸2b̸ifb̸byb̸seab̸b̸b̸"
> **TRIM longstring TO shortstring**
> will place "2b̸ifb̸byb̸sea" in SHORTSTRING

APPEND Command

APPEND provides the facility to join two or more strings together. The strings APPENDed together can be literals, variables, and/or strings created through the use of other DataFlex string commands (such as CHARACTER). String1 in the FORMAT below must always be a variable. This variable will contain the APPENDED string after the command has been executed. Strings cannot be APPENDed to a literal string constant.

FORMAT:

> **APPEND string1 string2 ...stringN**
> Where APPEND is the command, STRING1 is a string variable name, and STRING2 through STRINGN are literals or variables joined to STRING1. STRING1 remains after the command is executed as the variable name of the expanded string.

EXAMPLE:

> Where SIZE = "big"
> Where COLOR = " blue"
> **APPEND size color " balloon"**
> will result in the value of "big blue balloon" in SIZE

NOTE: If you wish to "collect" strings into a new, empty variable, it is necessary to clear that target variable with a MOVE "" to STRING1 command (substitute your variable's name for STRING1). If this is not done, the APPEND command will attempt (and fail) to operate on the string in its default initialized state, which is all spaces (to the declared length of the string). String variables are initialized in this way to reserve enough space in memory for the string's maximum size, but it cannot be APPENDed to in this state, since it is "full" of spaces.

UPPERCASE Command

The UPPERCASE command scans a designated string and converts all lower case letters to uppercase.

FORMAT:

```
        UPPERCASE string
        Where UPPERCASE is the command acting on STRING.
```

EXAMPLE:

```
        Where STRING="Smith"
        UPPERCASE string
        will return STRING as "SMITH"
```

CMDLINE Command

When executing a DataFlex configuration, command arguments entered at the operating system level as part of the initial command entry line can be stored to string variables for later repetitive use, with the CMDLINE command. This permits certain run-time choices to be made by the operator (or the menu system) and entered without separate prompting from within the configuration.

EXAMPLE:

```
        FLEX MYREPORT CON:
```

Executes the FLEX.COM file and runs the configuration called "MYREPORT". The "CON:" is not executed until a later point in the execution of the configuration when a CMDLINE command retrieves that argument and stores it to a string variable which is subsequently read by the configuration (and used, in this case, to direct the report's output to the screen).

CMDLINE will retrieve arguments with no limitation as to number from the last prior-executed operating system command or the last prior CHAIN command, whichever was executed more recently. Arguments are separated from each other by spaces (per the example above), and, once retrieved by CMDLINE, are no longer available from the same source, having been replaced by the next succeeding argument, if any.

FORMAT:

```
        CMDLINE string_arg
```

EXAMPLE:

```
        CMDLINE output_device
```

STRING OPERATIONS

The class (in-memory variable, database element, or window contents) of a string affects the way trailing blanks are handled. Basically, the differences are as follows (ƀ indicates a blank space):

Database Element: All characters plus additional blanks to fill the declared length of the element (set in File Definition). **Example:** "TheƀHagueƀ" (10 characters)

Window Contents: Only actual characters, including embedded blanks. **Example:** "TheƀHague" (trailing blank dropped)

In-memory Variable: Whatever was input to it. If a variable were loaded from the database element illustrated above, it would contain all 10 characters, including the trailing blank. If it were loaded from the window illustrated above, it would contain only the 8 non-blank characters plus the embedded blank.

Windows, then, TRIM strings just as does the TRIM command (see above), which is a way to TRIM leading and trailing blanks without MOVEing to, and back from, a window.

This is how to display "Miami,ƀFlorida" from two 10-character database elements, file.city and file.state:

```
STRING city 10
STRING state 10
TRIM file.city to city
TRIM file.state to state
SHOW city ",ƀ" state
```

This is how to display the same thing from in-memory variables var.city and var.state, each of which has length equal to their contents (5 and 7 respectively for Miami and Florida). The comma and space are handled differently only to illustrate an alternative; they could be handled either way in either instance:

```
STRING comma 2
MOVE ",ƀ" TO comma
SHOW var.city comma var.state
```

In general, it is preferrable to use in-memory variables for manipulation by the String commands.

CONDITIONAL EXECUTION

DataFlex provides two means for obtaining conditional execution of commands, INDICATORS and IF tests. Indicators will be discussed first. They are a special type of argument (often referred to as "flags") which conditionally controls command line execution. There are two possible statuses for an indicator: TRUE and FALSE.

An indicator is created with the INDICATOR command. (See the chapter on Declaring Data Elements.) Upon declaration, indicators remain in their default status of FALSE until an INDICATE command or some other event in the execution of the configuration resets it to the other status of TRUE.

If an indicator is TRUE, the command which follows it on a command line in a configuration will be processed. If an indicator is FALSE, the command line (or group) will be skipped, and execution will move to the next line. A TRUE indicator will also cause the processing of a WHILE - END, REPEAT - UNTIL or BEGIN - END structured command group which follows the indicator on a command line.

Once defined, an indicator may be used by enclosing it in brackets ("[" and "]") to identify it to the DataFlex command processor. It must be placed at the beginning of the command line(s) it is intended to control. The command processor recognizes the brackets, and does the appropriate conditional testing on the enclosed indicator.

FORMAT:

```
        [indicator_arg] command
```

EXAMPLES:

```
        [sale]          MOVE (invt.qty-amt_sold) TO invt.qty
        [member]        SHOW 'Enter membership code: '
        [credit_ok]     SHOW 'Credit approved up to ' cust.limit
        [single male]   SHOW 'Notice format for multiple indicators'
        [single][male]  SHOW "This format works the same!"
```

Using this format, the command processor tests the INDICATOR for a TRUE status. If the INDICATOR is TRUE, COMMAND is processed. If INDICATOR is not TRUE (FALSE), then the command is skipped, and the next command in the configuration is processed (or the next indicator is tested, etc.).

When creating a configuration, it is sometimes useful to be able to execute a command based on the FALSE status of an indicator rather than its TRUE status. This is done using the word NOT in front of an indicator on a command line to reverse the sense of the test applied to its status.

FORMAT:

```
        [NOT indicator] command
```

EXAMPLES:

```
        [a_test]        MOVE arg1 TO screen.1
        MOVES "arg1" if A_TEST is TRUE.

        [NOT a_test]    MOVE arg2 TO screen.1
        MOVES "arg2" if A_TEST is FALSE.

        [NOT member]    SHOW 'Who is this guy?'

        [NOT credit_ok] SHOW 'Confiscate the credit card!!'
```

Up to three indicators, separated by spaces, can be used together within the brackets to test whether a command line should be processed, or, if space allows, a set of brackets may be used for each indicator to improve readability. Either format works, and both are used in illustrations in this manual. A NOT can precede any indicator to reverse the sense of the test on the indicator which follows it. Concatenation of indicators in this manner interrelates them through "ANDs" (that is, all of them must be TRUE for a TRUE result--if any is FALSE, then the combination is FALSE).

FORMAT:

```
     [{NOT }indicator1 {NOT }indicator2 {NOT }indicator3] command
```

EXAMPLES:

```
        [sale][credit]  SHOW 'Enter item ID: '

        [credit sale]   SHOW 'Enter item ID: '
        Same function as example above.

        [sale][NOT credit]  SHOW 'Sorry, credit limit exceeded.'
```

Indicators can be used to conditionally set other indicators:

```
        [member]   INDICATE charge TRUE
```

SETTING INDICATOR STATUS

There are three distinct formats for using the INDICATE command to establish an indicator's status.

First, and most simply, indicator arguments can be explicitly set or reset using the following command format:

EXAMPLES:

```
        INDICATE indicator TRUE
        INDICATE indicator FALSE
```

Secondly, the TRUE or FALSE condition of an indicator can be established by the use of a comparison mode operator. There are eight (8) comparison modes for an indicator:

LT	less than
LE	less than or equal to
EQ	equal to
GE	greater than or equal to
GT	greater than
NE	not equal
IN	string inclusion
MATCH	string matching

When using the INDICATE command with a comparison mode operator, a status of TRUE or FALSE is given to the named indicator based on the outcome of the comparison of two arguments and the selected mode of comparison. That status is then tested each time that the indicator is encountered to determine whether the command line which follows the indicator in a configuration should be processed or not.

For example the following comparisons are:

<u>TRUE</u>	<u>FALSE</u>
5 GT 3	3 GT 5
666 EQ 666	666 EQ 999
999 LT 1000	1000 LT 999
'B' IN 'ABC'	'A' IN 'XYZ'
'ABC*' MATCH 'ABCDEF'	'A' MATCH 'B'

"IN" and "MATCH" have the following differences:

IN scans one string for the presence of another string; for example "A" is IN "DAC". When one string is IN another, the indicator is returned TRUE.

MATCH tests for one string equalling another. If the strings are equal the indicator is returned TRUE. It varies from the EQ comparison mode in that MATCH will perform the comparison against only part of the string and if it is equal to the same part of the other string the indicator will be returned TRUE. The partial string to be tested is represented with the asterisk convention used by most operating systems where "AB*" means anything that starts with "AB" is considered TRUE with whatever follows in the rest of the string not being evaluated or considered for the test. The asterisk may be used only in the first argument. The second argument must be a completely explicit string.

The comparison mode operator will be represented as "MODE" in examples in this manual.

The format of the INDICATE command with a mode operator creates an easy-to-read "sentence" which includes the INDICATE command, the indicator argument, and two other arguments which are compared by a comparison mode operator:

FORMAT
```
        INDICATE indicator_arg AS arg1 mode arg2
```

All of the illustrated components are required for a correct INDICATE command "sentence". The command "INDICATE" instructs the DataFlex command processor to set an indicator; the indicator_arg is the indicator which will receive the TRUE or FALSE status that is the result of the comparison; "AS" provides a syntax transition from the establishment of the indicator to the comparison itself; arg1 and arg2 must be present to create a valid comparison; and finally the comparison mode operator must establish what test is to be applied to arg1 and arg2 to arrive at the TRUE or FALSE status result to be applied to indicator_arg.

EXAMPLES:
```
        INDICATE of_age AS age GT 21

        INDICATE in_stock AS invt.qty GE order_amt

        INDICATE credit_ok AS cust.bal LT cust.cr_limit

        INDICATE eureka AS string1 IN string2

        INDICATE gotcha AS 'ab*' MATCH string2
```

The third format for setting an indicator uses other previously established indicators. To support this form of the INDICATE command, the following syntax is provided:

```
        GROUP, ANY, ALL, AND, OR and NOT
```

GROUP is used to define up to three other indicators which will be tested together to establish TRUE or FALSE status of the indicator defined by the INDICATE command. It replaces the transitional "AS" in the INDICATOR command definition "sentence".

ANY creates a TRUE status for the indicator argument if any of the up to three indicators included in GROUP is TRUE. ALL requires that all of the indicators in the GROUP be TRUE to pass a TRUE status to the indicator being set. The format for this form of INDICATE is:

FORMAT:

INDICATE ind GROUP ANY/ALL [ind ind {ind}] {AND/OR ANY/ALL [ind ind]}
The bracket characters, "[" and "]", used to enclose the GROUPed indicators are a necessary part of the command syntax. Each GROUP may contain up to 3 indicators, and up to 2 GROUPs (conjoined by AND or OR) may be in one command. Each group must be preceded by ANY or ALL.

EXAMPLES:

INDICATE us_car GROUP ANY [gm ford chrysler]
would set the indicator US CAR TRUE when any of the three indicators GM, FORD, or CHRYSLER was TRUE.

INDICATE conscript GROUP ALL [male citizen of_age]
would require that all three indicators MALE, CITIZEN, and OF_AGE be TRUE for the indicator CONSCRIPT to be set TRUE.

The AND and OR conjoiners allow conditional testing to be compounded to set an indicator. NOT enables the sense of an indicator to be reversed for a particular test.

EXAMPLES:

INDICATE eligible GROUP ALL [male singl of_age] AND ANY [rich handsom]
would set the indicator ELIGIBLE TRUE only if indicators MALE, SINGL, and OF_AGE and at least one of the indicators RICH and HANDSOM were TRUE.

INDICATE hunt GROUP ANY [squirrel possum] OR ALL [bear in_season]
would set the indicator HUNT TRUE if either of the indicators SQUIRREL or POSSUM were true, or if both BEAR and IN SEASON were TRUE.

INDICATE spanish GROUP ALL [cent_amer] OR ALL [south_amer NOT brazil]
would set indicator SPANISH TRUE if indicator CENT_AMER were TRUE, or if indicator SOUTH AMER were TRUE and indicator BRAZIL were FALSE.

The limitations on number of indicators in a group and number of groups in a command (3 and 2 respectively) can easily be overcome by use of successive INDICATE GROUP commands, and within a consecutive series of such commands, the same indicator can be reset many times to avoid consuming memory space for many different indicators.

PREDEFINED INDICATORS

Several predefined indicators are available as part of the DataFlex syntax:

FINDERR	TRUE if no record returned to buffer on FIND
FOUND	TRUE if record returned to buffer on FIND
SEQEOF	TRUE if end of sequential file
SEQEOL	TRUE if end of line in sequential file
KEYPRESS	TRUE if a key has been pressed (set by KEYCHECK)
PAGEBREAK	TRUE if new page (set by PAGECHECK)
MULTIUSER	TRUE if your DataFlex system is set multi-user

These indicators can be used at any point in a configuration to make portions of the configuration dependent on the condition which sets the status of the indicator. The status of the indicator can even be reset using the INDICATE command, but this should be done with great care, since other operations are likely to depend on internal settings of the status of these indicators.

IF TEST

The IF test is an alternate means of obtaining conditional execution. Indicators provide a means of applying the results of a test to numerous individual command lines throughout a configuration. The IF test, on the other hand, allows a specific test to be created on a command line that determines whether or not the single command following the IF test on the same line will be executed.

What is an IF test?

An IF test sets up a "comparison" sentence. In the sentence, a statement is made regarding the relationship of two items. They are compared using a comparison mode operator to see if they are equal, one greater than the other, or any of several other relationships.

EXAMPLE:

> **A car is bigger than a breadbox.**
> This sentence compares a car and a breadbox. Its comparison mode is "bigger". This sentence is TRUE.

The objective of the test is to arrive at a status result of TRUE or FALSE. For the purposes of conditionally executing a command line, if the command sentence correctly states the relationship between the variables, the sentence is TRUE and the command following the IF command will be executed. A FALSE sentence will pass by the command on the IF line, and go on to the command on the line following the IF line.

There are eight comparison modes for IF tests:

LT	less than
LE	less than or equal to
EQ	equal to
GE	greater than or equal to
GT	greater than
NE	not equal
IN	text string inclusion
MATCH	text string matching

When using the IF test, a status of TRUE or FALSE is created based on the outcome of the comparison of two arguments in the command line and the selected mode of comparison.

The following are examples of comparisons which might be stated in IF commands:

TRUE	FALSE
5 GT 3	3 GT 5
666 EQ 666	666 EQ 999
999 LT 1000	1000 LT 999
'B' IN 'ABC'	'A' IN 'XYZ'
'ABC*' MATCH 'ABCDEF'	'A' MATCH 'B'

Now all of the pieces are in place for the full command sentence.

FORMAT:

```
IF arg1 mode arg2 command
```

All of the illustrated components are required for a correct IF command "sentence". The command IF specifies the correct action to the command processor; "arg1" is a variable which will be compared to "arg2" according to the mode; "arg1" and "arg2" must be present because you cannot compare one thing; and the comparison mode operator must establish what test is to be applied; command will be executed if the test is TRUE.

EXAMPLES:

```
IF number1 GT number2 DISPLAY "Hooray!" TO screen.5
```

In this example if number1 is greater than number2, then the DISPLAY command will be executed, placing "Hooray!" in window 5 of pagename "screen" in the configuration. If number1 is smaller than number2, then there will be no DISPLAY.

```
IF (balance + purchase) LE credit_limit GOSUB sale
```

In this case, if the result of the expression is less than the credit_limit, then the configuration is sent to a subroutine called "sale". If (balance + purchase) exceeds credit_limit, the GOSUB will not be executed.

```
        IF payment LT balance BEGIN
           commands
        END
```

Here, the IF statement is being used to control the execution of a group of commands enclosed by BEGIN and END.

Most of the comparison mode operators are self-explanatory. However, "IN" and "MATCH" warrant further explanation. IN scans one string for the presence of another string; for example "A" is IN "DAC". When one string is present IN another, the test is TRUE.

MATCH tests for one string equalling another. If the strings are equal the IF test is TRUE. A special case exists for MATCH where only part of the string needs to be tested for equality for the status to be returned TRUE. This partial string match is represented by the asterisk convention used by most operating systems where "AB*" means to compare only the first two characters for anything that starts with "AB" regardless of the rest of the string. If the first two characters are "AB" the test is considered TRUE.

USING NOT

The "sense" of any IF test can be reversed by placing the word "NOT" in front of the first argument of the IF command:

FORMAT:

```
        IF NOT arg1 mode arg2 command
```

Point of interest...

When the DataFlex command processor encounters an IF command it internally expands the statement as a macro using the pre-defined indicator "LASTIF". A model/macro example is:

MODEL:

```
                IF xx GT yy command
```

MACRO:

```
                INDICATE LASTIF AS xx GT yy
      [LASTIF]  COMMAND
```

THIS PAGE INTENTIONALLY LEFT BLANK

STATUS File Indicator

Two forms of the STATUS indicator are available to indicate whether a record from a specified file is active in the buffer.

FORMATS:

> **INDICATE ind_arg STATUS filename_arg**
> The indicator IND_ARG is returned TRUE when the file is active (record is found), and FALSE if the record is not found.
>
> **IF STATUS filename_arg command**
> The command on the line is executed if the file is active (record is found), and not executed if the record is not found.

The STATUS indicator is used in a manner similar to the FOUND predefined indicator, but is useful in situations inolving successive FINDs in a number of files, because unlike the FOUND indicator, which is set according to the last FIND attempted, the STATUS indicator returns a result which is specific to the file which is named in the command.

CONTROLLING EXECUTION SEQUENCE

The Control command group provides structured flow of control within a configuration, error message handling, a debugging mode, and termination procedures.

Many of the control command group's actions alter DataFlex's default (top to bottom) sequence of configuration's command lines execution. Normally, commands are executed in the order in which they are written. Control commands are used to provide other sequences, in which, for example, often-used processes may be written once and "called" for execution from any of several other points in a configuration. The usual term for these called processes is "subroutines". Control commands can also provide conditional skipping of commands and groups of commands in a configuration. The "target" for these commands that tells DataFlex where to go and continue execution is called a "LABEL", and is defined as any continuous string of characters at the beginning of a line followed **immediately** by a colon (:). Labels must begin with a letter, which may be followed by up to 79 numbers and letters (no spaces or punctuation). These are examples of labels (they must be the **first** characters on their lines):

EXAMPLE:

```
LABEL:  command
EXAMPLELABEL:
command
...
command
```

A label may optionally be followed by a command on the same line (as in the first example above), or the following commands may be placed on following lines for readability (as in the second example).

GOTO Command

GOTO transfers the execution of a configuration to a labeled location in the configuration other than the next command line. Execution of command(s) following the "target" label will continue until the configuration ends, or another control command again transfers execution to another labelled location. As with all commands, the GOTO may be preceded by indicators

FORMAT:

```
GOTO label_arg
```

EXAMPLES:

```
GOTO update
GOTO del_rec
```

ON GOTO Command

The ON GOTO command transfers execution to a label that is part of a list of labels. The label is selected for execution based on the value of the integer variable or expression specified in the command. The value of the integer variable or expression will be used to count into the label list to an ordinal position designated by the value of the argument. **NOTE:** In the event the expression evaluates to less than 1 or greater than the number of labels in the list, the command will be ignored and the following command will execute. If this is an error condition in your application (should never happen), your configuration should declare an error (see ERROR command) on the next line.

FORMAT:

```
        ON integer_arg GOTO label_1 label_2 label_n
```

EXAMPLES:

```
        ON (items-144) GOTO apples, oranges, peaches, bananas
```
If ITEMS-144 equals 1, go to the label APPLES and execute from that point. If ITEMS-144 equals 2, go to ORANGES and execute from there. If the expression equals 3, go to PEACHES, for 4, to BANANAS, and otherwise go to the next line.

```
        ON screen.3 GOTO exit saveit deleteit
        ERROR 198
```
Go to labels EXIT, SAVEIT, DELETEIT according to the value in the window SCREEN.3. If SCREEN.3 has a value other than 1, 2, or 3, declare Error #198, where Error #198 has been defined for the circumstance in FLEXERRS.DAT

GOSUB Command

GOSUB transfers command execution to a labelled procedure elsewhere in the configuration with provision that when a RETURN command is encountered in the labelled procedure, execution will return to the command on the line **immediately following** the line on which the GOSUB appears. The RETURN command, of course, is written at the end of the called procedure. DataFlex supports nesting of up to 20 GOSUB procedures at one time (that is, a GOSUB which itself calls a GOSUB, and so on).

FORMAT:

```
        GOSUB label_arg
```

EXAMPLES:

```
        GOSUB my_routine
        GOSUB check_for_credit
```

ON GOSUB Command

The ON GOSUB command transfers execution to a labelled procedure whose label is one of a list of labels given in the command, with provision that when a RETURN command is encountered in the labelled procedure, execution will return to the command on the line **immediately following** the line on which the GOSUB appears. The RETURN command, of course, is written at the end of the called procedure. The label is selected for execution based on the value of the integer variable or expression specified in the command. The value of the integer variable or expression will be used to count into the label list to an ordinal position designated by the value of the argument. NOTE: In the event the expression has a value during running less than 1 or greater than the number of labels in the list, the command will be ignored and the following command will execute. If this is an error condition in your application (should never happen), your configuration should declare an error (see ERROR command) on the next line. DataFlex supports nesting of up to 20 GOSUB procedures at one time (that is, a GOSUB which itself calls a GOSUB, and so on).

FORMAT:

```
        ON integer_arg GOSUB label_1 label_2 label_n
```

RETURN Command

The RETURN command is used at the end of a labeled procedure called by a GOSUB command to cause the execution to transfer back to the origin of the control transfer. If RETURN has no argument, then execution will transfer back to the command line following the GOSUB command line which originally called the procedure.

RETURN can have an argument. Valid arguments for the RETURN command are other labeled commands in the configuration. This allows the chaining of labeled procedures within a configuration, or conditional branching to labeled procedures at locations other than the line following the calling GOSUB.

FORMAT:

```
        RETURN  // return to line after GOSUB
        RETURN label_arg   // return to a labeled command
```

EXAMPLE:

```
        [rainy] RETURN different_place
        RETURN
```

CHAIN Command

CHAIN terminates the execution of one configuration and begins the execution of the configuration named in the command argument, as if an operating system command were being executed. All data files (random and sequential) used in the configuration containing the CHAIN command are closed. All integer variables used by the terminating configuration remain available, with their values, for use by the succeeding configuration, after they have been redeclared in the succeeding one.

The argument of the CHAIN command may be either a literal string enclosed entirely within a pair of quotation marks, or a string variable. The literal contents of the argument may contain as many characters as your operating system will allow in a command (typically up to 128 characters, counting spaces). The literal string and the contents of the variable will be interpreted in the same manner: the characters preceding the first space will be considered the rootname of a DataFlex configuration, and DataFlex will append the .FLX extension and execute it. The characters following the first space and preceding the second will be retrieved by the first CMDLINE command in the configuration (see explanation of CMDLINE in this section). If there are characters following the second space, the second CMDLINE command (if any) will retrieve those characters up to the third space, and so on.

FORMATS:
```
        CHAIN "config_name {string ... string}"
        CHAIN string_var
```

EXAMPLES:
```
        CHAIN "inventory"  //Execute configuration INVENTORY.FLX
        CHAIN next_config  //Execute first substring in variable
                           //NEXT_CONFIG and pass following substrings
        CHAIN "accts_rec LST: cust_name"  //Execute configuration
                                          //ACCTS REC.FLX and pass
                                          //substrings LST: CUST_NAME
```

CMDLINE Command

When executing a DataFlex configuration, command arguments entered at the operating system level as part of the initial command entry line can be stored to string variables for later repetitive use, with the CMDLINE command. This permits certain run-time choices to be made by the operator (or the menu system) and entered without separate prompting from within the configuration.

EXAMPLE:
```
        FLEX MYREPORT CON:
```
Executes the FLEX.COM file and runs the configuration called "MYREPORT". The "CON:" is not executed until a later point in the execution of the configuration when a CMDLINE command retrieves that argument and stores it to a string variable which is subsequently read by the configuration (and used, in this case, to direct the report's output to the screen).

CMDLINE will retrieve arguments with no limitation as to number from the last prior-executed operating system command or the last prior CHAIN command, whichever was executed more recently. Arguments are separated from each other by spaces (per the example above), and, once retrieved by CMDLINE, are no longer available from the same source, having been replaced by the next succeeding argument, if any.

FORMAT:

```
        CMDLINE string_arg
```

EXAMPLE:

```
        CMDLINE output_device
```

DEBUG Command

DEBUG is a mode command which toggles on and off from its default condition of "OFF". When on, it will cause the line number in the compiled .FLX configuration file being executed to display on the screen as it is being executed whenever the configuration is run. Constants and variables will also be displayed as they are declared if the DEBUG command is the **first** command in the configuration. The intermediate FCODE output of the DataFlex Macro Processor is also displayed with the line number when a configuration containing a DEBUG command is compiled. Naturally, this command would not normally be left in a finished configuration ready for running by operators.

ERROR Command

Error reporting is done using the ERROR command. The trapping of the error must be done prior to the ERROR command execution. System errors are trapped by DataFlex. The system errors (those under 100) are listed in the appendix under "RUN TIME ERRORS", and may also be read directly from the DataFlex database file FLEXERRS.DAT. Errors which occur during execution must be trapped by the configuration.

The argument of the ERROR command is the number of the error which is to be declared, specified either literally in the command or by the use of a(n INTEGER) variable. The errors are defined in the DataFlex FLEXERRS.DAT file supplied with the system. Custom error messages may be added to the file by use of the DataFlex configuration FLEXERRS, using error numbers greater than 100. The added error message(s) will then be displayed on the screen on the last line if the ERROR command is ever executed with the number of the message (on an error condition). Such error message(s) will then be "standard" to all DataFlex configurations run with the FLEXERRS.DAT file.

FORMAT:

```
        ERROR #
```

EXAMPLE:

```
        ERROR 4
        would display the message "SEEK TO UNWRITTEN EXTENT (CP/M
        ERROR)."
```

CLEARWARNING Command

Warnings and messages printed on the last line of the CRT using the ERROR command can be removed with CLEARWARNING. It prints spaces to the line and deletes whatever was displayed there. CLEARWARNING is automatically executed after an ACCEPT command. It has no argument.

FORMAT:

```
        CLEARWARNING
```

ABORT Command

ABORT returns control to the DataFlex Menu. Any open data files are closed when ABORT is executed. However no "cosmetic" cleanup is done, so at least a CLEARSCREEN would be in order in most cases when this command is executed from a screen-oriented configuration. ABORT has no argument.

FORMAT:

```
        ABORT
```

BLOCKS AND LOOPS

The DataFlex Structured Control commands provide a means of executing parts of a configuration conditionally and/or in loops. The parts of a configuration under control of Structured Control commands are called "blocks", and they are made up of a sequential series of commands which are placed between the "beginning" command (e.g., BEGIN, REPEAT, WHILE, FOR), and the "ending" command (END, UNTIL, LOOP). Indicators and parameters of the commands control whether, and how many times, the blocks are executed.

BEGIN - END Command

The simplest of the structured control commands is BEGIN - END. The BEGIN defines the start of a group of commands that are executed until an END command is encountered. Indicators and "IF's" can be used to test both BEGIN and END commands. The BEGIN and END commands do not support arguments. After execution of the line containing the END command, execution goes to the next line of the configuration.

FORMAT:

```
        BEGIN
          COMMAND GROUP
        END
```

EXAMPLE:

```
[found] BEGIN
           DISPLAY list.name TO screen.1
           DISPLAY list.address TO screen.2
           DISPLAY list.city_state TO screen.3
        END
```

If [FOUND] is TRUE, the DISPLAY commands will all be executed. If it is FALSE, they will be skipped.

REPEAT - UNTIL Command

The REPEAT - UNTIL construct executes a group of commands in a loop UNTIL a specified condition is met or created, inclusive. If the condition is already met at the time execution first encounters the loop, the loop will still be executed at least once.

FORMAT:

```
        REPEAT
           commands
        UNTIL arg mode arg
```

EXAMPLE:

```
        REPEAT
           MOVE (count+1) TO count
           SHOW count
        UNTIL count EQ 5
        The loop would REPEAT until COUNT equalled 5 (inclusive).
```

REPEAT - LOOP Command

The REPEAT - LOOP construct is a loop which requires that execution continue until a conditional statement (indicator or IF command) within the command group or on the LOOP command line terminates the processing and causes an exit.

FORMAT:

```
        REPEAT
           commands
              conditional or indicator-controlled exit command
           commands
        LOOP
```

EXAMPLES:

```
                REPEAT
                   SHOW "*"
                   INPUT "Enter data: " some_data
[KEY.ESCAPE]       ABORT
[KEY.RETURN]    LOOP
                Loop only if RETURN key is pressed.  Abort if ESCAPE
                key is pressed.


                REPEAT
                   CALCULATE (count + 1) TO count
                   IF count GE 100 GOTO a_routine
                LOOP
```

WHILE - END Command

The WHILE command establishes a test like an IF command which is executed until the test becomes FALSE. The END establishes the end of the command group, after execution of which the process checks the WHILE condition, and if it is TRUE, loops back to the beginning of the group, and if it is FALSE, continues on with execution of commands following the END statement. If upon first execution, the "IF" condition is FALSE, the command group will be skipped (not executed at all).

FORMAT:

```
        WHILE arg_1 mode arg_2
           commands
        END
```

Where arg_1 is a variable, mode is an IF comparison mode (see discussion of IF command), and arg_2 is a string, date or numeric value, or a variable whose value is reset within the loop.

EXAMPLE:

```
        WHILE in_stock GT 0
           INPUT sale
           CALC (in_stock - sale) TO in_stock
        END
```

FOR - FROM - TO - LOOP Command

This is a loop construct that has a specified range for its operation. It is based on execution for a certain number of repetitions defined by starting (FROM) and ending (TO) integer limits. If execution first encounters the loop while the value of the integer is **outside** the limits in the command, the command group will **still** execute once before exiting.

FORMAT:

```
        FOR integer_var FROM arg1 TO arg2
           commands
        LOOP
```

Where integer_var is any previously-declared integer variable, and arg_1 and arg_2 is are numeric constants, expressions, or variables.

EXAMPLES:

```
        FOR timer FROM 1 to 500
           MOVE timer TO an_integer
        LOOP
```

All BEGIN, LOOP, and END commands in the Structured Control command group can be used with indicators or IF commands for multi-level conditional execution. Structured Control commands should not be used in the ENTRY sections of the ENTER and ENTERGROUP macros, because the BACKFIELD key is capable of defeating the initial command which controls entry to the loop.

DIRECT CONTROL OF THE CONSOLE

The Console Input/Output (I/O) command group provides a facility for directly addressing the CRT device (terminal screen) and to perform a range of CRT operations which are sometimes useful in the development of more advanced applications. Before any of the console I/O group of commands is used, it should be considered whether it might be easier or more direct to utilize DataFlex's powerful screen imaging facility which is available at the beginning of any and every configuration. The console I/O command group is generally less efficient than screen images for moderate to elaborate displays, and is intended for use only in exceptional situations.

The actions of the Console I/O command group are totally independent of the particular codes and routines which activate screen functions on any particular computer or terminal. DataFlex acquires and maintains system-specific data about your CRT when the SETSCREEn utility is run. The data from SETSCREEn is stored in the file FILELIST.CFG. Data about your system is read from FILELIST.CFG each time DataFlex is initialized to provide the necessary system-specific data for the execution of the commands.

SHOW Command

The SHOW command provides the facility to output data to the CRT device. It will not output to the printer even if output is directed to the printer (see WRITE and WRITELN commands). The data output to the CRT may be of any type (constant, variable, numeric, string, expression, etc.). The SHOW command acts on an argument which defines the data that is to be "shown" on the CRT. Literal arguments must be embedded in single or double quotation marks, and produce a display of the argument itself. Arguments without quotes are variables, and produce a display of the contents of the variable. Multiple arguments can be used. No carriage return is output at the end of execution of a SHOW command. Two consecutive SHOW command lines will output data to the same line, one after the other, left to right, on the screen. Literal strings to be SHOWn cannot be longer than 80 characters.

FORMAT:

```
        SHOW argument {argument ... argument}
```

EXAMPLES:

```
        SHOW name ", " address ", " city'
```

Display the value of the string variable NAME, followed by a comma and a space, then the value of the variable ADDRESS, followed by a comma and a space, followed by the value of the variable CITY on the screen at the cursor position.
(continued next page...)

EXAMPLES (continued from preceding page)

```
SHOW "Hello, " pers_name
```
Display Hello, followed by the contents of the variable PERS_NAME on the screen at the cursor position.

```
SHOW "Input today's date (MM/DD/YY): "
```
Display Input today's date (MM/DD/YY): on the screen at the cursor position.

```
SHOW (12.70+TOTAL)
```
Display the results of adding 12.70 to the value of the variable named TOTAL on the screen at the cursor position.

SHOWLN Command

This command is the same as SHOW, except that SHOWLN outputs a carriage return after displaying all of the arguments that it outputs to the terminal, giving the cursor a new line on which to display, including succeeding SHOWLN commands. It is useful, among other things, for displaying vertical columns of output on the screen, which is usually done in a loop.

FORMAT:

```
SHOWLN argument {argument ... argument}
```

EXAMPLE:

```
WHILE area_code LT 500
        SHOWLN "Area Code " area_code "processing completed"
LOOP
```
would display successive lines on the screen reading "Area Code <number> processing completed" until the value of AREA_CODE equalled or exceeded 500, where <number> represents the value of the individual area code.

INPUT Command

The INPUT command accepts keyboard input and assigns the data received to a variable, named in the command argument. Data can be INPUT **only** from the keyboard, and by no other means. Once the INPUT data is in the variable, it can be acted upon or processed under the variable name within the configuration. INPUT functions with all valid variable types, and if the input data does not match the type of the variable named in the argument, an error will be declared (e.g., if an INTEGER type variable is named and alpha characters are input from the keyboard, an error will occur). After entry of the data, any Flex-Key will cause it to be loaded, although the RETURN key is the usual key to use for the purpose.

FORMAT:

```
INPUT 'optional prompting string' var_arg
```

EXAMPLES:

> **INPUT "Enter the date (MM/DD/YY): " todays_date**
> Display Enter the date (MM/DD/YY): at the current cursor position, pause execution, and store subsequently-keyed response under the variable TODAYS DATE
>
> **INPUT quantity**
> Halt execution and store keyboard input under the variable QUANTITY.

GOTOXY Command

GOTOXY sends the CRT cursor to a position on the screen specified in the arguments to the command, providing control of the location of the actions of other console I/O commands on the CRT device. GOTOXY provides no other function except positioning; other commands **must** be used to do something after the cursor has been moved to the desired location.

FORMAT:

> GOTOXY arg1 arg2
> Where arg1 is the LINE and arg2 is the COLUMN where the cursor is to be placed on the CRT.

EXAMPLES:

> GOTOXY 0 0
> Position the cursor at the extreme upper left (HOME) of the screen (line 0, column 0).
>
> GOTOXY 12 40
> Position the cursor on line 12, column 40 (exact center of a 24-line, 80-column screen).

The valid range for values of the line and column arguments is determined by the system configuration as defined in SETSCREEN.

CLEARXY Command

The CLEARXY command clears from the coordinate screen position specified in the command argument to the end of the screen. After execution, the cursor is positioned at the argument coordinates. This command **must not be used within a named PAGE, as it will cause an exit from whatever PAGE it is invoked in.**

FORMAT:

> CLEARXY arg1 arg2
> Clear the screen from Line arg1, column arg2.

EXAMPLES:

```
        CLEARXY 0 0
        Clear the entire screen (see CLEARSCREEN below).

        CLEARXY 13 0
        Clear from the beginning of line 13 to the bottom of the
        screen (bottom half of a 24-line screen).
```

CLEARSCREEN Command

CLEARSCREEN clears all printable characters from the screen and sends the cursor to the "HOME" position. No argument is used with this command.

FORMAT:

```
        CLEARSCREEN
```

KEYCHECK Command

When a KEYCHECK command is executed, a predefined indicator, KEYPRESS, is set TRUE if a key was pressed at any time since input was last accepted to a window or an ACCEPT statement. (The character generated by whatever key was pressed is discarded.) KEYCHECK is useful primarily for interrupting long processing cycles and reports. In such uses, it is recommended to display a prompt to the screen such as "PRESS ANY KEY TO INTERRUPT". KEYCHECK may be followed on the same line with another command which is executed if KEYPRESS is TRUE (a key has been pressed), and skipped if KEYPRESS is FALSE (no key has been pressed). The KEYPRESS indicator may be used at any time after execution of a KEYCHECK command (to set its state), so that intervening commands may be executed unconditionally before the command whose execution depends on the state of KEYPRESS. KEYCHECK does not distinguish among keys; it responds the same to any key which generates a character (it does not respond to CTRL, SHIFT, or BREAK).

FORMAT:

```
        KEYCHECK {optional following command}
```

EXAMPLES:

```
        KEYCHECK                    //Set KEYPRESS to TRUE or FALSE.
        commands                    //to execute unconditionally
        [KEYPRESS] command          //executes depending on KEYPRESS

        KEYCHECK GOSUB shutdown
        If a key has been pressed, execute subroutine SHUTDOWN.
```

NOTE: KEYCHECK commands should not be used under the REPORT macro. If it is desired to control branching by keypresses in the REPORT macro, the appropriate subroutine under the name RPT.KEYPRESS should be defined in the configuration, ending with the special form of RETURN command for the REPORT macro, RETURN RPT.LOOP.

INKEY Command

INKEY inputs one character from the keyboard into an argument. It does **not** display the character on the screen. It does halt execution of the configuration until a key is pressed, and is very useful for this purpose. The command requires an argument; it cannot be used without one.

FORMAT:

```
        INKEY argument
```

EXAMPLES:

```
        INKEY a_key
        Store the first key input to the keyboard under the
        variable A_KEY.
```

SCREENMODE Command

The SCREENMODE command sets the high- and low-intensity screen attributes as established in the DataFlex SETSCREEn utility. The attribute will remain until another SCREENMODE command is executed, or a PAGE is displayed, at which time the attribute will be set to low intensity. SCREENMODE does not affect display of a PAGE. There are no arguments for this command other than the two shown below:

FORMAT:

```
        SCREENMODE 1    sets high intensity

        SCREENMODE 2    sets low intensity
```

THIS PAGE INTENTIONALLY LEFT BLANK

MANIPULATING RECORDS

The DataFlex Database commands provide the facility to find, save and delete records in the database, and to establish "relationships" between and among records.

The power of these individual DataFlex commands far exceeds that available in conventional high level languages, making "Flex" configurations smaller, more consistent and easier to maintain than alternative application development techniques.

Database Commands and FILEDEF

The DataFlex Database commands are integrally linked to data dictionary information established with the DataFlex FILEDEF and AUTODEF utilities. These utilities create a dictionary of specifications for a data structure and stores it on disk. The data dictionary contains data on each file's number of fields, field length and type, decimal formatting on numeric fields, name including drive reference, multi-user status, key index structure, etc. All other DataFlex operations can then "look up" the file specifications in the data dictionary whenever needed. This precludes the necessity for each program to maintain file-specific data and substantially reduces application development time.

Understanding Record Buffers

The data in your DataFlex files resides on the disk drives. One ACTIVE record from each file can reside in RAM memory. The place in memory where this record resides is called the RECORD BUFFER. When a record is read from the disk to the record buffer, data can be extracted from and/or changed in the buffer. The record can then be written back to disk (SAVED). The buffer is where DataFlex does its work on a given record.

An ACTIVE record buffer contains a record that has been previously read from the database. That is, the record exists both on disk and in main memory in exactly the same form. If changes have been made to an active record buffer, that record is considered EDITED. An edited record MUST be saved back to the database in order to retain the changes which have been made. An edited record is still active.

An INACTIVE record buffer contains either no data or data that does not (as yet) exist in the database (specifically, newly entered data which has not been SAVEd to disk). A CLEAR record buffer is an inactive record that has had no data PUT into it at all. In a CLEAR buffer all numeric and date fields are zero, and all ASCII fields contain spaces. When an inactive but not clear record buffer is SAVED, a new record will be created.

FILE MANIPULATION

When using the highest level of the DataFlex commands, like ENTER and REPORT, all of the details of the record buffers, FINDing, SAVEing and CLEARing are taken care of for you. The way in which these operations are done is completely transparent to the operator and the configurator. But when you have a requirement that necessitates manipulating the files manually, you must understand the basics of file manipulation. When using the manual file commands, records are NOT automatically related, saved, and cleared. You must ensure that these operations are done, and done in the proper sequence.

There are four basic manual operations to perform on a database:

1) FINDING:	Read records for reference only.
2) EDITING:	Read records, modify them, and write them back to the database.
3) CREATING:	Create new records.
4) DELETING:	Delete existing records.

The procedure for FINDING a record is:

a) CLEAR the record buffer.
b) Put the KEY data into the record buffer.
c) FIND the record by the KEY field.
d) Optionally, RELATE the found record.

After a FIND you may get data out of the record buffer. If the FIND is unsuccessful, the KEY data will be in the buffer and the record buffer will be INACTIVE.

The procedure for EDITING an existing record is:

a) FIND the record (see above) to edit.
b) Put new data into the record buffer fields to edit the record.
c) SAVE the record.

A SAVE must be done after a "put" to an active record buffer to move the data to disk. Continuing processing after editing an active record without SAVEing it will cause Error 82, and the file will have to be REINDEXed.

The procedure for CREATING a record is:

a) CLEAR the record buffer.
b) Put the data into the record buffer fields.
c) SAVE the record.

The procedure for DELETING a record is:

a) FIND the record (see above) to delete.
b) DELETE the record.

OPEN Command

An OPEN command makes all database files associated with a file "df_filename" (data and key files) available for DataFlex operations. (See the FILEDEF documentation for information on df_filename). Specifically, when a file is OPENed, the referenced data file, and all associated key index files are opened, buffer space is automatically allocated in memory, and upon completion of the OPEN operation, all files are available for on-line processing.

The OPEN command(s) should be at the beginning of the configuration and **must be executed before the files associated with the df_filename** can be addressed by other commands.

FORMAT:

OPEN file_name_arg {index.n}
Where OPEN is the command, "file_name_arg" is the "SHORT NAME" (created in FILEDEF) of the file to be OPENed, and n represents the number of an index that may optionally be loaded into a memory "buffer" to speed sequential access to records in a file.

EXAMPLE:

```
OPEN inventory
OPEN personnel index.2
```

In the second example above, the "index.2" option after the df_filename "personnel" elects to load the second index for that database file into a memory buffer.

The optional index buffering only provides a speed improvement when the access to records is sequential, as in a report. Random operations, such as transaction entry to non-sequential records, will not benefit from buffering.

The usefulness of the index buffering option is, as with most other options, subject to tradeoffs. The number of the index is established in the index definitions as previously created in the FILEDEF function. If the option is selected, record locations required by FIND and other accessing operations are read directly from the computer's active memory (RAM) rather than via a separate (slower) disk access and file-read operation.

The tradeoff when using record buffering is the sacrifice of a portion of the computer's RAM that is consumed by this process. The overall size of your configuration, number of files opened, RAM capacity of your computer, and so on, determines how much RAM is available over and above what is required to carry out the application's basic functions, and will dictate whether the index-buffering option is worthwhile, or even possible.

In a multi-user environment, the optional index buffering is allowed and DataFlex will maintain indexes properly, even in those cases where buffered indexes are updated.

When DataFlex configuration source files are being compiled, a special file needed by the compiler macro processor must be resident on the logged device on which the compiler is located. The needed file is referenced as DF_FILENAME.FD. "DF_FILENAME" is established in the FILEDEF utility when the file is defined. Its name is maintained separately so that the ".FD" file can be addressed without assuming the drive specification attached to the data file.

FIND Command

The FIND command is used to get a desired specific record from a data base on disk and place it into the record buffer. FIND uses data placed in the record buffer (normally by prompted keyboard input) to search a key index, "find" a match and get the record. For example, in a file maintenance situation, the operator could be prompted for a record ID or "key" of some kind, an account number, part code, etc. Data entered in response to the prompt is placed into the buffer and a search for a match (or other relationship--see MODES below) is performed by the FIND command.

The FIND command must reference an existing index or record number. All segments in the index will be used to locate the record with the FIND operation, including record number, if included in the index.

FIND must be given a parameter by which to execute. Do we want to search for a record with an exact match to the "key", or do we want the record with a "key" greater than the value entered to the buffer? This parameter is called the MODE of the FIND. Five MODEs are available:

LT	Less Than
LE	Less than or Equal to
EQ	Equal
GE	Greater than or Equal to
GT	Greater Than

Finally, we must define what index is to be used for the FIND operation. This is specified by the number of the index to be used, as defined in the database (see File Definition) as INDEX.n, where n is the index number.

FORMAT:

```
FIND mode filename BY INDEX.n
```

Where n is a number between 1 and 5 (for 8-bit computers) or between 1 and 10 (for 16-bit computers). INDEX.BATCH and RECNUM may be used alternatively. The index must first have been defined in the database definition.

EXAMPLES:

```
FIND EQ invt BY INDEX.2
```

Find a record in the INVT file by INDEX number 2 with a key equal to the data in the buffer.

```
FIND GE member BY INDEX.3
```

Find a record in the MEMBER file by INDEX number 3 where the value is equal to or greater than the data in the buffer.

NOTE: The filename in the command must previously have been OPENed with the OPEN command.

The FIND GE is probably the most used command/MODE combination. It will retrieve a record based on an exact match with data in the buffer, or if no match is found, the next record following the value of the buffer data.

The result of executing a FIND operation is shown by either of two indicators: FINDERR and FOUND. FINDERR will be TRUE when the FIND operation is not successful by the specified MODE. FOUND will be set TRUE when a record is found according to the specified MODE. These indicators can be used to direct the action of the configuration based on the outcome of the FIND command execution.

An alternate form of the FIND command can be used where the index number is not known.

FORMAT:

```
FIND mode filename.fieldname
```

EXAMPLE:

```
FIND GT INVENTORY.DESCRIP
```

The filename.fieldname must, of course, be indexed. If it is not, an error will occur at runtime when the command is executed.

The FIND command works only when it occurs within a definite sequence of commands, specifically:

1. CLEAR the record buffer.
2. MOVE the finding data (data to be matched or related to).
3. Execute the FIND.
4. Execute conditionally on successful or unsuccessful FIND.

The following commands represent an example of this sequence:

```
INPUT "Enter the name you wish to find:  " find_name
CLEAR people
MOVE find_name TO people.lname
FIND EQ people.lname
[FINDERR] GOTO no_match
```

CONFIGURATION EXAMPLE:

```
/*                                      // starts configuration

        STRING pause 1                  // declare variable PAUSE

        OPEN maillist                   // opens maillist file

START:  CLEARSCREEN                     // clear the screen

        CLEAR maillist                  // clear before FINDing

        INPUT "Enter NAME to display: " maillist.name
                                        // get keyboard input

        FIND GE maillist BY INDEX.1     // search for match

[FOUND] BEGIN           // FOUND is set TRUE automatically by a
                        // successful FIND operation
                        // Commands from here to the next END are
                        // executed only if the FIND was successful

                SHOWLN maillist.address

                SHOWLN maillist.cty st_zip

        END
                        //  ****  ERROR TRAP ****
                        // FINDERR is set TRUE automatically by an
                        // unsuccessful FIND operation
                        // The next command is executed only if the
                        // FIND attempt was unsuccessful (no match).

[FINDERR]       SHOWLN "No names matching: " maillist.name

        INPUT "PRESS RETURN TO FIND ANOTHER, <ESC> TO QUIT" PAUSE

[KEY.ESCAPE]    ABORT

        GOTO START
```

SAVE Command

The DataFlex SAVE command takes data in a record buffer and "saves" it on disk. The command can have mulitiple arguments on the command line. If multiple files are OPEN and relationships exist, all files related to the file being saved will be automatically ATTACHed (see below) and saved.

FORMAT:

```
        SAVE filename_arg1 {filename_arg2 ... filename_argn}
```

EXAMPLES:

```
        SAVE inventory
        SAVE client transaction system
```

NOTE: The filename(s) in the command must previously have been OPENed with the OPEN command.

The action of the SAVE command varies based on how data has been placed in the buffer, and what has happened to it once it arrived there.

An empty (or CLEAR ...see below) buffer that has data moved into it from any source (window, another file), will simply move the data from the buffer into a newly created record on disk.

If a record is FOUND (past tense of FIND), and the data in it is then modified (e.g., by keyboard entry) then the buffer (containing the altered data) will be SAVEd back to the same record in which it was originally stored.

NOTE: It is important to remember that the process of editing in DataFlex occurs when data is moved to the record buffer. Therefore, no changes are made to the database on disk until a SAVE command is executed after the modification of the data in the buffer. It is an error to edit a record and not SAVE it.

SAVERECORD Command

SAVERECORD saves a record from its buffer to the disk. No automatic ATTACH is executed and no related files are saved; only the file that is the argument of the command is saved.

FORMAT:

```
        SAVERECORD filename_arg
```

NOTE: The filename(s) in the command must previously have been OPENed with the OPEN command.

CLEAR Command

The CLEAR command acts upon the record buffer of the filename referenced in the command argument(s). Any data in the buffer is flushed by the execution of a CLEAR.

FORMAT:

```
        CLEAR filename_arg1 {filename_arg2 ... filename_argn}
```

EXAMPLES:

```
        CLEAR myfile
        CLEAR myfile yourfile ourfile
```

NOTE: The filename(s) in the command must previously have been OPENed with the OPEN command.

The CLEAR command should be used at any time that an initialized record buffer is required. For example, if keyboard entry has been used to create data in the buffer for a FIND, and no record is found (FINDERR is TRUE), then a CLEAR would purge any entries that had been made to the buffer before making fresh entries in an attempt to proceed and FIND other records. If, on the other hand, you wish to de-active an active (FOUND) record and retain entered data in the buffer, a MOVE 0 TO RECNUM command will accomplish that.

DELETE Command

The DELETE command removes an existing record from the data base. A record must be FOUND before it can be DELETEd. When the DELETE command is executed, all key indexes for the record being deleted are updated on-line to remove key entries for the deleted record.

FORMAT:

```
        DELETE filename_arg {filename_arg2 ... filename_argn}
```

EXAMPLES:

```
        DELETE names notes reminders
```

NOTE: The filename(s) in the command must previously have been OPENed with the OPEN command.

RELATE Command

The RELATE command works in conjunction with the FIND command. After a record has been found with FIND, RELATE will find all related records in the database. For example, if a transaction record is found and placed in the buffer, the execution of a RELATE command could find a related master record(s) in the database and place it (them) in the buffer additionally.

FORMAT:

```
        RELATE filename_arg
```

EXAMPLES:

```
        FIND GE invt_trans.posting  //executed first
        RELATE invt_trans
```

NOTE: The filename(s) in the command must previously have been OPENed with the OPEN command.

A FIND must be executed before RELATE. The record buffer for the filename_arg must be "active" for RELATE to function. Files to be used as the argument(s) of the RELATE command must be OPENed prior to the command execution. Related files that are not OPENed are ignored by the RELATE command. In the case of a "chain" type relationship, an attempt to RELATE a file that is not OPENed precludes access to other files further up the chain.

The use of RELATE assumes that the proper relationships have been established in FILEDEF between (among) the files.

ATTACH Command

When file relationships have been established in FILEDEF, the ATTACH command provides the means of placing the data on which the relationship is based into a new database record. If it is desired only to SAVE related records, it is not necessary to ATTACH them, since ATTACHment is done automatically in execution of the SAVE command. Therefore, ATTACH should be used only where further processing is required before the SAVE is done.

FORMAT:

```
        ATTACH filename_arg
```

EXAMPLE:

```
        ATTACH transactions
```

NOTE: The filename(s) in the command must previously have been OPENed with the OPEN command.

Upon execution of the ATTACH command, DataFlex moves the record number or field data that is defined as the basis for the relationship into the new record. All related files that are open when the command is executed will have a relationship created.

The ATTACH command only moves data into the new database record. That data, and the entire new record, is not a permanent part of the database until a SAVE command is executed to put the information on disk. ATTACH moves data from "higher" to "lower" data files only within an application data structure. Briefly, a "higher" file is one with records (such as "employer") which are related to many other records (such as "employee"), and a "lower" file is one which contains those many records which relate to the "higher" one. These terms are more fully discussed in the section of this manual on File Definition - Step One - Plan the Relationships Among the Data Files.

ZEROFILE Command

The ZEROFILE command is potentially very destructive of data, and should be used only with great caution. Its effect is to **permanently** erase all data from a database file. It may be useful to do this in order to remove experimental or sample data from a set of database files (without erasing the files themselves and having to redefine the entire database), or to reinitialize a database for a new period.

FORMAT:

```
        ZEROFILE filename
```

NOTE: The filename(s) in the command must previously have been OPENed with the OPEN command.

Again, improper use of this command on valid, needed data can be disastrous. Great caution is advised in using this command.

SYSTEM FILES

It is frequently desirable to have a file with only one record to hold today's date, your company's name and address, the last-used invoice number, cumulative totals, and other system-oriented information. This type of file is referred to as a SYSTEM FILE.

A system file is defined, created and changed just like any other DataFlex file, but system files can contain only one record.

To define a system file, set the "MAXIMUM NUMBER OF RECORDS" in the "SET SYSTEM PARAMETERS" of FILEDEF to one (1) when the file is set up. DataFlex recognizes this parameter, and automatically reads the single record from this special type of file into a buffer without a FIND opration. Any data in the system file is then available in the buffer without a disk access.

System files should not have any indexes since there is only 1 record.

When used with the ENTER command, system files must be OPENed, but the system file should not be included on the ENTER command line.

An example of a system file can be found in the DEMO files as SYSFILE. Invoice uses the date from SYSFILE.

It is recommended that file one be used as the system file. Also, that File 1, Field 1 be assigned as the company name and that File 1, Field 2 be the date. Append other system file fields to the end of these as needed. There may be multiple system files.

MANIPULATING FILES

The System command group provides for the execution of operations that would normally be done at the operating system command level. These commands can be executed within a DataFlex configuration file as required. In that these commands are very powerful, and can make major alterations to a system, they should be used with respect and care. Supportive operator prompting and verification prior to the execution of these commands are recommended.

A sub-group of commands wihtin this group manipulates the operating system directory. Filename arguments in the sub-group that are literals must be enclosed in single or double quotation marks. Those filenames that are referenced in data elements (string variables or ASCII windows) do not require quotes.

ERASEFILE Command

Deletes the file named in the command argument from the operating system directory.

FORMAT:

```
        ERASEFILE filename_arg
```

EXAMPLES:

```
        ERASEFILE "myfile.dat"

        ERASEFILE file_to_kill
```

RENAME Command

Changes the name of a file in the operating system directory.

FORMAT:

```
        RENAME filename_arg1 TO filename_arg2
```

EXAMPLES:

```
        RENAME file_to_rename TO "TEMP.DAT"

        RENAME oldfile TO newfile
```

COPYFILE Command

Transfers data from one file to another, or transfers file(s) from one device to another. "Wild card" or ambiguous file name entries are supported for the purposes of this command so that groups of files can be manipulated.

FORMAT:

```
        COPYFILE sourcefile(s) TO destfiles(s)
```

EXAMPLES:

```
        COPYFILE "a:current.dat" TO "b:history.dat"

        COPYFILE "a:*.flx" TO "b:"

        COPYFILE screen.6 TO 'b:newstuf.txt'
```

DIRECTORY Command

Displays a directory of the disk files specified in filespec_arg.
FORMAT:

```
        DIRECTORY filespec_arg
```

EXAMPLES:

```
        DIRECTORY "c:*.dat"

        DIRECTORY screen_window.20
```

FILELIST Command

The FILELIST command displays information from the list of current DataFlex data files. It can read the PATHNAME, DISPLAY NAME, DataFlex file name and file number from FILELIST.CFG. The file number is stored in a pre-defined integer variable called FILENUMBER. There are three variations on the use of the command.

FORMAT 1:

```
        FILELIST filenum_arg TO string1_arg {string2_arg}
```

Where filenum_arg is the number of the desired file, string1_arg is where the file's user display name is stored and/or displayed, and string2_arg is where the file's DataFlex file name is displayed or stored.

EXAMPLE 1:

```
        FILELIST 22 TO screen.6 screen.7
```

Puts the file's user display name for File 22 into Window 6 on page "screen", and its DataFlex filename into Window 7.

The FILELIST NEXT command scans FILELIST.CFG and returns the next ACTIVE file (the next larger number for an existing active file). The file number is returned in FILENUMBER. The value of FILENUMBER is the starting point for the execution of the command. To get a complete FILELIST, initialize FILENUMBER to zero (0).

FORMAT 2:

```
        FILELIST NEXT TO string1_arg {string2_arg}
```

EXAMPLE 2:

```
        MOVE 0 TO FILENUMBER
        REPEAT
                FILELIST NEXT TO list.2 list.3
        UNTIL FILENUMBER EQ 0
```

Would sequentially output the entire list of display names and DataFlex file names for all active files to Window 2 and Window 3 of Screen list.

The FILELIST PATHNAME command outputs the pathname and (optionally) DataFlex filename of the file currently pointed to by FILENUM.

FORMAT 3:

```
        FILELIST PATHNAME TO string1_arg {string2_arg}
```

EXAMPLE 3:

```
        FILELIST PATHNAME TO screen.15 screen.16
```

SYSTEM Command

The SYSTEM command returns control of the computer to your operating system. All files opened in the configuration will be closed in the process of the executing the command. No CRT screen "clean-up" is performed by SYSTEM, so any cosmetic work must be done by other commands. SYSTEM differs from the ABORT command in that ABORT will return to the DataFlex Menu if it is on the disk drive, while SYSTEM explicitly exits to the operating system.

FORMAT:

```
        SYSTEM
```

CAUTION: If DataFlex is being run under control of a batch or submit operating system file, SYSTEM will cause the next entry in the batch file to begin executing, just as any other return to the operating system would do. If the "RETURN TO OPERATING SYSTEM" prompt which occurs upon use of the SYSTEM command is confusing to operators, an explanatory prompt just before the SYSTEM command can help alleviate such confusion.

RUNPROGRAM Command

The RUNPROGRAM command runs an operating system level program (.COM or .CMD FILE). It can have two arguments. Argument One is the program name, Argument Two is the command line to pass to the program. After the program has executed, control is returned to the operating system.

FORMAT:

```
        RUNPROGRAM prog_name.com_arg prog_arg
```

EXAMPLES:

```
        RUNPROGRAM "PIP" "B:=A:*.*"

        RUNPROGRAM screen.2 screen.3

        RUNPROGRAM "edit" prompt_input
```

REGISTRATION Command

The REGISTRATION command will retrieve the registered name and serial number of a DataFlex license and place the data into variables that can be used in a configuration.

FORMAT:

```
        REGISTRATION string_arg numeric_arg
```

EXAMPLE:

```
        REGISTRATION licensee_name serial_number
```

Use of the REGISTRATION command to display the name of the application author (company or individual) is effective in identifying the software to the creator/owner, and is one of the main purposes for which this command is provided.

SYSDATE Command

The SYSDATE command retrieves date and time information from the operating system, provided the date and time have been properly set in the operating system. The SYSDATE command stores the system date to the first variable named after the command, the hour (in 24-hour format) to the second (optional) argument, and the minutes to the third, if a third argument is specified in the command. The date will be returned in DATE format, and the second two will be returned as two-digit numbers. The name DATE should not be used, since that is a

DataFlex reserved word.

FORMAT:

```
        SYSDATE <date_arg> {hour_arg} {minute_arg}
        Where <date_arg> is a date variable, and {hour_arg} and
        {minute_arg} are optional 2-digit numeric variables.
```

The command returns the year in two digits (as in 12/24/44). If a four-digit format is desired (12/24/1944), it can be obtained by adding the amount 693975 to the returned value.

EXAMPLE:

```
        SYSDATE sdate hour minute
        MOVE (sdate+693975) TO sdate
```

All MS-DOS-based operating systems and operating systems which support the MP/M (BDOS 105) DATE system call are supported. CP/M 2.2 and CP/M-86 1.0 do not have time and date functions. Among operating systems which are supported are: CP/M 3.0, Concurrent CP/M, MP/M-86 and MS-DOS, as well as TurboDOS, DPC/OS 3.0, MS-DOS, and NStar.

THIS PAGE INTENTIONALLY LEFT BLANK

DIRECT INPUT AND OUTPUT

The Sequential I/O command group provides a means of reading data from, and writing data to, physical input/output devices (such as printers) and disk files produced by, or for further processing by, programs other than DataFlex (WordStar, CBASIC, etc.). These commands do not use "image" formatting. Data sent to files and devices from this command group must be formatted (where necessary) in the configuration to satisfy the requirements (delimiters, etc.) of the destination device or non-DataFlex program.

NOTE: The first four commands of this group use an underline character (_) within the command name. This character is integral with the command name and cannot be omitted without causing a compile-time error.

DIRECT_INPUT Command

DIRECT_INPUT specifies a disk file name or device from which sequential input is to be received for processing by a DataFlex configuration. The argument for the command is the name of the file or device, or the name of a variable or window which contains the desired file name or device designator. If the argument is the literal name of the file or device designator, it should be surrounded by quotation marks. If the argument is a string variable, quotation marks should not be used. Execution of the command automatically opens the necessary buffers and makes the file or device fully ready for input operations. If a specified file or device cannot be found in the system, the end-of-file indicator is set TRUE. If there is need in the application to declare an error in this case, an error message can be established which is triggered by the state of the end-of-file indicator.

FORMAT:

> **DIRECT_INPUT string_arg**
> Where string_arg contains the source file name or device designator.

EXAMPLES:

> **DIRECT_INPUT "wsmerge.txt"**
> Open a file named WSMERGE.TXT for input
>
> **DIRECT_INPUT "RDR:"**
> Open the port designated RDR: in the operating system for input
>
> **DIRECT_INPUT screen.1**
> Evaluate the contents of Window #1 on the screen and open the file or device identified thereby

DIRECT_OUTPUT Command

DIRECT_OUTPUT identifies a disk file or physical device to which sequential data is to be sent from a DataFlex configuration. The argument for the command is the name of the file or device. Execution of the command automatically opens the necessary buffers and makes the file or device fully ready for output operations. If the argument to the command is the literal name of the file or device, it should be embedded in quotes ("). If, on the other hand, the argument is the name of a variable which contains the file name or device designation, quotes should not be used. If a file or device is specified in the command which does not exist in the system, a new disk file will be created under the name specified in the command. CAUTION: If the command names an existing file, that file will be erased by the DIRECT_OUTPUT command, with no warning to the operator unless one is explicitly written into the configuration.

FORMAT:

DIRECT_OUTPUT string_arg
Opens the file or device specified by the contents of the string variable named STRING_ARG.

EXAMPLES:

DIRECT_OUTPUT "myfile.txt"
Opens for output the file named MYFILE.TXT.

DIRECT_OUTPUT screen.4
Get name of device or file from Window 4 on Page SCREEN.

CLOSE_INPUT & CLOSE_OUTPUT Commands

These two commands simply close the file or device opened by the DIRECT_INPUT AND DIRECT_OUTPUT commands respectively. They operate on any device or file which may be open, and require no argument.

FORMAT:

```
CLOSE_INPUT
CLOSE_OUTPUT
```

READ Command

The READ command reads sequential data from a file or device and assigns data thus read to the variable name(s) given in the argument of the READ command. READ recognizes the comma (,) as the delimiting character between data elements, and ignores commas which are contained within double or single quotation marks. If there are single quotes (') in embedded material, double quotes (") should be used as the embedding character. If there is need for double quotes, single quotes may be used for embedding. It is not recommended to attempt to embed any single data element containing both types of characters.

EXAMPLE:

```
        Ring,Mahoney and Arner,Miami,Florida
```

would be interpreted as four data elements, specifically:

```
        1:  Ring
        2:  Mahoney and Arner
        3:  Miami
        4:  Florida
```

while:

```
        "Ring, Mahoney and Arner",Miami,Florida
```

would be interpreted as three data elements, as follows:

```
        1:  Ring, Mahoney and Arner
        2:  Miami
        3:  Florida
```

The READ command will read as many data elements as there are arguments given in the command, unless the CR/LF end-of-record character is encountered in the source data before all arguments have been assigned data elements.

FORMAT:

```
        READ argument1 ... argumentN
```

EXAMPLES:

READ an_element
Read one element from the current input line and store it under variable name AN_ELEMENT.

READ name address city state zip
Read the first data element in the record and store it under the variable name NAME, read the next element and assign it to ADDRESS, the next to CITY, and so on.

If the number of data elements in a record exceeds the number of arguments in a READ command, execution will stop short of the end of the record, after reading the last data element for which there is an argument to assign it to. Thus, if the second command example shown above were run against this data:

```
        Data Access,129th Terr.,Miami,FL,33156,(305)987-6543<CR>
```

only the first five elements (name, address, city, state, zip) would be read into DataFlex. The phone number would not be read (although a succeeding READ command could, and would, read it). If the number of READS for a record exceeds the number of fields in the record, the exceeding READS output null fields for the output record.

Thus, if the second command example above were run against this data:

```
        Data Access,129th Terr.,Miami,FL<CR>
```

The resultant data in DataFlex would be:

```
        NAME:  Data Access
        ADDRESS:  129th Terr.
        CITY:  Miami
        STATE:  FL
        ZIP:  (blank)
```

READLN Command

The READLN command is similar to the READ command (see above), except that it reads all data from wherever the command is invoked to the end of the record (defined by a carriage return (CR) character in the source data). Like the READ command, argument(s) set the name(s) of the variable(s) under which read data will be stored.

Where a READLN command is issued with one argument, all data from the source file or device on a line is stored under a single variable having the command argument as its name. When multiple arguments are used, data elements will be stored sequentially under the argument names, provided that the source data is delimited by commas (as it customarily is in sequential files created under MBASIC*, CBASIC*, WordStar* MailMerge*).

FORMAT:

```
        READLN argument1 argument2 ... argumentN
```

EXAMPLE:

```
        READLN input_string
```

Read a line of data and store it under the variable name INPUT_STRING (regardless of delimiters which may be within the line).

```
        READLN name address city state zip a/c phone
```

Read a line of data and store the first data element under variable name NAME, the second under ADDRESS, the third under CITY, and so on.

The multi-argument command READLN A B C would execute in exactly the same fashion as the following group of READ commands:

```
        READ A
        READ B
        READLN C
```

Where the last command causes a READ from the third data element to the next <CR>, inclusive.

If the number of data elements on a source line happens to exceed the number of arguments in a READLN command, the "exceeding" data elements will be stored under a variable together with the last element for which there was an argument. The variable name for this last (group of) data element(s) will be the last argument in the READLN command. In this situation, the (comma) delimiters between each pair of "exceeding" data elements will also be stored in the resultant data element.

Generally, multi-argument READLN commands will be used where a fixed number of data elements constitutes one record, as where:

```
        Name,Address,City,State,Zip<CR/LF>
```

represents a record. The READLN command which would read this data in under the appropriate variable names would be:

```
        READLN Name Address City State Zip
```

It would be necessary in this instance that any empty data element be represented by a delimiting comma, as in:

```
        2nd National Bank,,Anytown,Anystate,12345
        Where the address element is blank or non applicable.
```

SEQEOF and SEQEOL Predefined Indicators

DataFlex provides two predefined indicators, SEQEOL (sequential end of line) and SEQEOF (sequential end of file), which can be used to control operation of READing and other configurations. Both indicators are set FALSE upon execution of a READ or READLN command, and are reset TRUE, respectively, upon the READing of an end-of-line character (<carriage return>), and upon the READing of an end-of-file character (^Z). NOTE: READLN will always read to, and including, the next end-of-line character, but will not read an end-of-file character WITH the last record if there is an EOL immediately ahead of it. It will have to do one more READLN (creating an empty record) to detect the EOF. Similarly, READ will not READ an EOL with the last field if the EOL is immediately preceded by a comma; it will require an additional READ (or READLN) to detect the EOL. These factors should be kept in mind in the design of READing configurations, record counters and the like, as should the exact format of your source data.

WRITE Command

The WRITE command sends data sequentially to the device or file name given in a preceding DIRECT_OUTPUT command. Arguments to this command identify the variable names under which the data is found for output. Multiple arguments can be used with this command. No carriage returns or delimiters are provided by use of this command, for which reason they must be added to the data in the argument to the command as literals (see examples below). If it is desired to include quotation marks (", also called double quotes) in the literal, most non-DataFlex programs permit the use of single quotes (') as the embedding character. This provision affords total flexibility in adding delimiting characters to data being output.

FORMAT:

> **WRITE argument**
> Output one data element stored in the variable ARGUMENT.
> **WRITE arg1 arg2 ... argN**
> Write the first data element from variable ARG1, the second from ARG2, and so on through ARGn.

EXAMPLES:

> **WRITE "datadata"**
> Output the literal string datadata
>
> **WRITE "Your balance is: " cust.bal_due " as of " due_date**
> Output the literal Your balance is: followed by the value of the variable cust.bal_due followed by the literal as of followed by the value of the variable due_date.

It is important to note that all formatting, spacing and punctuation for the output information must be supplied by the WRITE Command.

EXAMPLES: Where FIRM.NAME = Dewey, Cheatum and Howe

> **WRITE firm.name**
> would output Dewey, Cheatum and Howe
>
> **WRITE '"' firm.name '"'**
> would output "Dewey, Cheatum and Howe"
>
> **WRITE '"' firm.name '",'**
> would output "Dewey, Cheatum and Howe",

WRITELN Command

The WRITELN command sends data sequentially to the physical device or file name specified in a preceding DIRECT_OUTPUT command, just as in the WRITE command. Just like the WRITE command, WRITELN outputs data under the variables and/or literals specified in the command argument. The difference is that WRITELN outputs a carriage return (CR) after data for the last argument on the command line has been sent. This character signifies the end of a record in many non-DataFlex programming languages and devices, such as WordStar* Mailmerge*, CBASIC* and MBASIC*. WRITELN can have multiple arguments on the command line and, just as with the WRITE command, it is necessary to include any needed delimiting characters as literals in the command line.
FORMAT:

WRITELN argument
Output the contents of variable ARGUMENT, followed by a <CR>

WRITELN arg1 arg2 ... argN
Output the contents of ARG1, then ARG2, and so on through ARGn, the last followed by a <CR> character.

EXAMPLES: where arg1 = apples, arg2 = peaches, and arg3 = pumpkin pie

WRITELN arg1
Would output apples<CR>

WRITELN arg1 "," arg2 "," arg3
Would output apples,peaches,pumpkin pie<CR>

THIS PAGE INTENTIONALLY LEFT BLANK

USING FUNCTION KEYS

DataFlex provides a set of system-independent function keys to perform numerous entry and search functions, as described in the section on the SETSCREEn procedure. They are referred to as DataFlex Command Keys, or simply "Flex-Keys". On terminals lacking function keys, the Flex-Keys must be assigned to control key sequences. On terminals having function keys, they can be assigned to the various function keys available on the keyboard. In either case, the name and function of the key is maintained over all systems and applications.

It is very important to note that the Flex-Keys have NO intrinsic functions. They derive their various functions from the DataFlex configuration which is running, either from KEYPROC commands discussed in this section, or from the use of the ENTER or ENTERGROUP command macros discussed respectively in other sections of this manual.

The Flex-Keys are divided up into two main groups: input-terminating Flex-Keys and field editing Flex-Keys. The field editing keys include backspace, the arrow keys, insert character, and delete character, and all have effects on the contents of a window or other means of keyboard entry. The input-terminating keys, on the other hand, control movement from field to field, and any of them functions in the manner of the <RETURN> key to enter any characters which may have been entered into the active window at the time the input-terminating Flex-Key is pressed.

THE INPUT-TERMINATING FLEX-KEYS

INDICATOR	FUNCTION
KEY.RETURN	RETURN OR ENTER
KEY.ESCAPE	EXIT THE CURRENT FUNCTION
KEY.FIELD	BACK UP ONE WINDOW OR FIELD
KEY.FIND	FIND A RECORD
KEY.SFIND	SUPERFIND
KEY.SAVE	SAVE A RECORD
KEY.DELETE	DELETE A RECORD
KEY.CLEAR	CLEAR THE FORM OR SCREEN
KEY.NEXT	GO FORWARD ONE RECORD
KEY.PREVIOUS	BACK UP ONE RECORD
KEY.UP	UP ARROW
KEY.DOWN	DOWN ARROW
KEY.USER	FUNCTION DETERMINED BY CONFIGURATOR
KEY.HELP	DISPLAY HELP SCREEN

Note that the field edit keys are not listed here since they function normally under the ACCEPT command.

When the ENTER macro command is used, all the Flex-Keys are automatically endowed by the macro with their customary functions (see list of the Flex-Keys above). Where the ENTER macro is used, it is still sometimes desirable to alter the default functions of the Flex-Keys. Where the ENTER macro is not used, the Flex-Keys have no effects at all, and they must be provided in the configuration by use of either of the two methods described below. The effect(s) of using the Flex-Keys can be altered or extended by two different means: (a) use of the pre-defined indicators which are automatically controlled by use of the Flex-Keys; and (b) use of KEYPROC subroutines.

THE USER-DEFINED FLEX-KEY

No function is assigned automatically to the User-Defined Flex-Key. The key is provided so that the configurator may have the option of providing any chosen function to be available with a single keystroke. Such special function can be provided only through a Key Procedure (see below) within one or more configurations.

USING THE FLEX KEY INDICATORS:

Each time one of the field-terminating Flex-Keys is used, the predefined indicator relating to that key is set TRUE, and all the other Flex-Key indicators are set FALSE. For example, when the <RETURN> key is used to terminate entry to a window, the indicator KEY.RETURN is set TRUE. These indicators, which are listed above, all have the prefix "KEY." followed by a distinguishing term which refers to the default function of the key. Subsequent actions or branching may be controlled by use of the reset indicators.

EXAMPLE:

```
        GETNAME:          ACCEPT COMPANY.NAME
        [KEY.ESCAPE]      ABORT
                          ACCEPT COMPANY.ADDRESS
        [KEY.ESCAPE]      ABORT
        [KEY.FIELD]       GOTO GETNAME
```

This example opens the configuration for input to the variable COMPANY.NAME, with the label GETNAME: at the beginning of the example. The second line, beginning with the indicator reference [KEY.ESCAPE] provides that, if the ESCAPE key is pressed, execution of the function will cease (ABORT). If other than the ESCAPE key is pressed, the input is accepted and execution goes on to open for input to the variable COMPANY.ADDRESS. The following line, again beginning with [KEY.ESCAPE], provides the same ABORT result of use of the ESCAPE key. The next line ([KEY.FIELD]) provides that if the BACK FIELD key is used during or after input, execution will return to the first step of this example (GETNAME:). If neither key is pressed, execution goes on to other commands following the example.

USING KEY PROCEDURES:

Using the key indicators can be practical in configurations involving relatively little input, but in configurations involving a large number of ACCEPT or INPUT statements, it will normally be preferable to standardize the actions of the Flex-Keys. Key procedures (activated by use of the command KEYPROC) provide the means of programming function key effects uniformly across an entire configuration for whenever a particular key is used. Like general-purpose subroutines (see GOSUB command in the chapter on Control Commands), KEYPROC subroutines must be terminated with either a RETURN command to resume execution of the configuration at the command following the ACCEPT or INPUT statement which called the KEYPROC, or an ABORT command to terminate execution of the configuration.

FORMAT:

```
        KEYPROC flex_key
```

Where flex_key is one of the Flex-Key indicators listed above. When the Flex-Key which relates to the indicator given in the command argument is pressed, execution will automatically branch to the KEYPROC subroutine.

EXAMPLES:

```
        KEYPROC KEY.ESCAPE
        CLEARSCREEN
        ABORT
```

Causes use of the ESCAPE key to clear the screen and terminate execution of the configuration.

```
        KEYPROC KEY.CLEAR
        CLEARFORM
        RETURN
```

Causes use of the CLEAR ALL key to clear the form and return to execution of the next command of the configuration.

If it is desired that a key procedure not be in effect under certain conditions during execution of a configuration, indicator-driven conditionals can be included within the KEYPROC subroutine to provide the desired conditional effect.

EXAMPLE:

```
                    KEYPROC KEY.ESCAPE
        [NOESCAPE]  RETURN
                    ABORT
```

This example illustrates the provision that pressing the ESCAPE key will cause execution to ABORT unless the indicator NOESCAPE is set TRUE, in which case execution RETURNS to the next command in the configuration (the ESCAPE key is prevented from "working").

There are some other commands that can ONLY be used within key procedures. These commands can alter the place the key procedure returns to and allow key procedures to do things like trap an error and return to the original window:

ENTAGAIN Command

ENTAGAIN stands for enter again and will make the key procedure return to the same window and ACCEPT command that called it. If a key procedure determines that there was an error it can then go back to the same place. ENTAGAIN does not use arguments.

FORMAT:

```
        ENTAGAIN
```

EXAMPLE:

```
                        KEYPROC KEY.ESCAPE
        [NOESCAPE]      ENTAGAIN
        [NOESCAPE]      RETURN
                        ABORT
```

In this example, it is provided that the ESCAPE key trigger an execution ABORT (first and last lines), unless the NOESCAPE indicator is set TRUE by action of other parts of the configuration, in which case execution returns to the place at which the ESCAPE key was pressed, and the cursor goes back to the beginning of the window it came from.

ENTERMODE Command:

The ENTERMODE command establishes the begining of a series of data entry commands and serves to set up the correct modes for the next few commands. The ENTERMODE command should always be the first command in a series of ACCEPT commands when the BACKFIELD (see next command) is used:

BACKFIELD Command

The BACKFIELD command will cause the key procedure to return to the ACCEPT command PREVIOUS to the one that was just executed. This makes the cursor seem to back up one window. The BACKFIELD command will not go back past an ENTERMODE command. A properly-placed ENTERMODE command should be used since otherwise the BACKFIELD command will go back as many command lines as is necessary to find an ACCEPT to execute. The BACKFIELD command can be controlled by the status of indicators. This command should be used judiciously in configurations containing loops, GOTOs, GOSUBs and the like, since the command in effect executes a GOTO to the previous ACCEPT command, and this may seriously upset processing if it occurs in a loop. The operator may be prevented from making such an error by manipulation of an"inverse" user-named indicator (NOBACK, for example) included within the KEYPROC to disable the key at the appropriate points in execution. This technique is illustrated above for the ESCAPE key.

EXAMPLE:

```
        KEYPROC KEY.FIELD
        BACKFIELD
        RETURN
```

This example merely defines the function of the BACK WINDOW (KEY.FIELD) key to back up one window.

HELP KEY:

The key is called KEY.HELP and is generally assigned to ^Q on the standard keyboard.

The HELP command can be used within a standard key procedure and is implemented in ENTER and ENTERGROUP. The standard format for this key procedure is:

```
         KEYPROC KEY.HELP
           HELP
           ENTAGAIN
         RETURN
```

HELP Command

The help command with no arguments will go to the next sequential page after the currently displayed page and see if the word "HELP" is on the pagename line ("/" line). If so, that page will be displayed as a help screen. Pressing the help key again will display the next help screen (if available). Pressing any other key will return to the original page and window.

If the HELP command is used with an argument of a page name, that page will begin the help screen sequence.

EXAMPLE:

```
/SCREEN1
        INPUT   SCREEN

                NAME: ___________________________________

/SCREEN1HELP    HELP

Please enter your name in the blank provided.

Press any key to continue
/SCREEN2

        ADDRESS: ____________________________
                 ____________________________

/SCREEN2HELP    HELP
Now, please enter your home address.

Press any key to continue, or press HELP for more help

/SCREEN2HELP2   HELP
If you don't know your home address, try getting out your driver's
license and copying the address off of it.

Press any key to continue
/*
```

MULTI-USER FUNCTIONS

DataFlex provides true multi-user operation. This means multiple users with simultaneous write privilege to the same record at the same time. This provides a transparent multi-user operation with data protection to the field level. No other microcomputer product can match DataFlex in the multi-user environment.

Problems inherent in the "traditional" method of handling multi-user operations (record lock on each read) are illustrated by the following example:

A company has a large number of customers, say 20,000, but only sells 10 different items. Traditional locking on customer records would seldom produce a visible lock, but if one of 10 inventory part records was locked on READ, the system would become virtually unusable with only two or three operators. DataFlex allows unlimited READ access to the database and RE-READs the data when the operator SAVES a changed record.

In DataFlex, the record or system is locked only for the time of the SAVE. Updates are then done based on the fresh, safe, data. And only those fields that have actually been changed are saved to the record, thus preserving any other changes. This type of multi-user is virtually undetectable to the operator, greatly in contrast to a read-time lock.

Multi-user updates are of two basic types: transactional and replacement. Transactional updates are generally numeric adds or subtracts from totals in a master record. Since the update is based on the re-read (safe) data, updates of totals will always be accurate. Replacement updates, (changing an address or credit limit) do not usually require multi-user interaction, but do require that the application not arbitrarily rewrite fields that were not changed as traditional methods provide. DataFlex keeps track of which fields were changed in order not to rewrite an old value over changes made by another user subsequent to the first user's READ.

Here is an example of how this works:

Three users have the inventory part "ABC" on their screens. The quantity in stock (10) and the description (WIDGET) are shown on each screen. User 1 is invoicing the sale of one widget, while User 2 is invoicing a sale of two. User 3 is changing the description to "WIDGET, GREEN". All three users read their (identical) screens, key their input, and then press "SAVE" within the same second. Let's say that the actual sequence of SAVES fell out as User 1 first, then User 2, and User 3 last.

User 1's SAVE will re-read the record and find it unchanged. His sale of one widget will then be subtracted from the quantity in stock, and the quantity in stock written back as 9.

User 2's SAVE will then take control, reread the record (9 in stock), subtract two, and write back the proper number (7).

User 3's SAVE will then get control, reread the record (with 7 in stock), ONLY update the description, and write it back correctly. All this occurs with speed which typically is imperceptible to any user.

In such a system, replacement privilege to transactionally-updated fields should be very carefully limited to responsible, thoroughly-trained users. To make sure two users do not sell quantities in excess of the quantity in stock, the application configuration should be written to check for amount sold exceeding quantity in stock (and declare an error while rejecting the sale).

The foregoing provisions reduce the time a system or record is locked to the very brief interval actually required for the SAVEing of a record instead of the reading and entry speed of the operators. Since one field does not interfere with another and transactional updates are optimally handled, the statement "DataFlex provides transparent multi-user protection to the field level" is amply justified.

MULTI-USER OPERATION UNDER THE ENTER MACRO

Multi-user operation is provided automatically within the ENTER macro. An ENTRY command will only update the database if data in one or more windows has been changed (unless the FORCEPUT option is in effect). Also, the active records are reread from the disk BEFORE the save or delete procedures. If it is desired to re-display the buffer data after REREAD but before SAVE, the ENTDISPLAY command should be used for the purpose. During the save and delete procedure, only one terminal will have write privilege, and all other updates initiated during the processing interval will be delayed until that update is done. For these reasons, the following rules must be followed when using ENTER in a multi-user environment:

1) When in the ENTRY section, refer to the data windows instead of the record buffer, since the record buffer is not changed until the record is SAVEd.

2) When in a SAVE or DELETE procedure, refer to the record buffer instead of the data windows, since data in the record buffer could have changed during a re-read.

3) NEVER allow data entry of any kind within a SAVE or DELETE procedure since other users will be locked out of SAVE operations while the first operator is in the procedure.

MULTI-USER COMMANDS:

The following commands will only function in multi-user systems. There is no harm in providing them in single-user systems for compatibility, since they have no effect in single-user mode.

LOCK Command

The LOCK command will prevent other users from LOCKing the database, so that only one user at a time has database write privileges. Once LOCK is invoked, no other user may update the database until the invoking user's procedure is complete. The LOCKed condition should be kept in force for only those succeeding commands for which it is truly required.

UNLOCK Command

The UNLOCK command will restore the system to normal state after a LOCK or REREAD. It is imperative that the system be UNLOCKed after a LOCK or REREAD is issued.

FOR EVERY REREAD OR LOCK THERE MUST BE AN UNLOCK!

REREAD Command

The REREAD command first LOCKs the system and then re-reads all of the ACTIVE records in the database. Since this is done in a LOCKed condition, the resultant re-read records are "safe" for update. Use of this command requires that all updates (edits) to an active record buffer are done at one time, since the LOCKed condition should only be kept in force for those few commands for which it is required.

In the locked condition resulting from a LOCK or a REREAD, data entry statements such as INPUT, ACCEPT, ENTRY or INKEY should NEVER be used in a configuration.

The REREAD command can optionally be followed by a list of files to REREAD. In this form of the command, only those files listed will be REREAD from among the open data files. Without the option, all open data files for which there is data active in the buffer will be re-read.

IFCHANGE Command

In a multi-user environment it may be desired to have certain commands execute conditionally based on whether there has been a change in a window. The command to do this is IFCHANGE. IFCHANGE is followed by a window number and the command which is to be executed if the window has changed.

EXAMPLE:

```
        IFCHANGE form.1 MOVE form.1 TO invt.id
        If the operator has changed window one, MOVE it to INVT.ID
```

The IFCHANGE command can also respond to change in the record buffer. If the argument is a file name, then the command after it will only execute if the record buffer has been changed since the last FIND.

SETCHANGE Command

The SETCHANGE command creates a "change" in the window without altering any data. It satisfies the conditions required for execution of the IFCHANGE command unconditionally.

FORMAT:

```
        SETCHANGE window
```

DESPOOL Command

The DESPOOL command will cause the spooler to start printing if a spooling function is provided in the operating system.

FORMAT:

```
        DESPOOL
```

EXAMPLE OF UPDATE IN MULTI-USER ENVIRONMENT:

```
        FIND INVT BY INDEX.1       // Get the record to update
        DISPLAY INVT.ID TO FORM.1 // display default data
        DISPLAY INVT.PRICE TO FORM.2
        ACCEPT FORM.1              // accept changes to default data
        ACCEPT FORM.2
        REREAD                     // prepare for multi-user update
                                   // update only changed fields
        IFCHANGE FORM.1 MOVE FORM.1 TO INVT.ID
        IFCHANGE FORM.2 MOVE FORM.2 TO INVT.PRICE
        SAVE INVT                  // save the file
        UNLOCK                     // and unlock
```

FIELDINDEX AND WINDOWINDEX

FIELDINDEX and WINDOWINDEX provide means of accessing and processing ranges of fields in a data file or of windows on a screen, affording greater efficiency in both the writing and execution of configurations for sequentially accessing ranges of fields and windows.

WINDOWINDEX is a predefined variable which counts windows in screen image PAGEs, and FIELDINDEX similarly counts fields in databases. Where WINDOWINDEX and FIELDINDEX start counting is controlled by the placement of an ampersand (&) character which is appended, in the case of a window, to a window number, and in the case of FIELDINDEX, to a field number. For example, if the current value of WINDOWINDEX is 3, SALES.1& means "the third window after window SALES.1," or the fourth window in the PAGE named SALES. If the value of WINDOWINDEX is changed to 4, SALES.1& would refer to the next, or fifth, window in the PAGE. Likewise, if FIELDINDEX were equal to 1, EMPLOYEE.LNAME& would refer to the first field after field EMPLOYEE.LNAME. The ampersand cannot be used by itself as a field number or window number. It must be appended to an actual window name or number, or to an actual field name (as defined in FILEDEF) or number.

The following example would SHOW the contents of fields 0 through 20 of the record currently in the buffer:

```
FOR FIELDINDEX FROM 0 TO 20
        SHOW EXAMPLE.RECNUM&
LOOP
```

The initial FOR statement increments FIELDINDEX from zero through 20 with each repetition of the loop. The SHOW command takes the whichever record from the EXAMPLE data file is currently in the buffer and shows RECNUM (the record number--always the first, or 0th, field in any record) and the 20 fields which follow it, on the screen, through the presence of the & after the field name RECNUM. On the second pass, RECNUM& evaluates as "the first field after the record number", and so on through 20.

CAUTION: When using FIELDINDEX and WINDOWINDEX to MOVE data, make certain that the data type of the destination fields or windows is the same as that of the field (or window) name in the command line to which the ampersand is appended (RECNUM in the example above). FIELDINDEX and WINDOWINDEX check the type of this data element and assume that the destination windows or fields will be of the same type. If a mismatch exists between the data types, it will produce a BAD FORMAT IN EXPRESSION error at run time. If source data types are mixed (as when the field referred to is RECNUM, a NUMERIC type, and those following are ASCII fields), use of one or more of the type-changing forms of the MOVE command will overcome the problem. The commands (MOVENUM, MOVEINT, and MOVESTR) are described under Element Processing Commands, and should be used according to the data type(s) of the destination fields or windows.

The steps for using WINDOWINDEX and FIELDINDEX, then, are:

1. Identify the field.# or window.# at which the range of windows or fields you want to process begins. You can start and stop where you like, but you can't skip fields or windows;

2. Write the processing command(s) you want for either field.#& or window.#&, and

3. Set up a loop or other counter to drive FIELDINDEX or WINDOWINDEX, whichever is appropriate, through the range you wish to process.

In another example, we could have a page (display screen) called SALES with the twelve calendar months on it as windows displaying totals by month, and two entry windows at the bottom for the entry of additive amounts and of the number of the month to which the entered amount is to be added. The month numbers shown on the screen display below are strictly to cue the operator--they are not used by the system. Likewise, the window numbers shown are merely for explanatory purposes. They would not normally be displayed on the screen:

```
/SALES

        # MONTH            SALES

 1    January            $______.__ (window 1)
 2    February           $______.__ (window 2)
 3    March              $______.__ (window 3)
 4    April              $______.__ (window 4)
 5    May                $______.__ (window 5)
 6    June               $______.__ (window 6)
 7    July               $______.__ (window 7)
 8    August             $______.__ (window 8)
 9    September          $______.__ (window 9)
10    October            $______.__ (window 10)
11    November           $______.__ (window 11)
12    December           $______.__ (window 12)

  MONTH #: __. (window 13)  AMOUNT TO ADD: $_____.__ (window 14)
/*

ACCEPT SALES.13
CALC (SALES.13-1) TO WINDOWINDEX
ACCEPT SALES.14
CALC (SALES.14+SALES.1&) TO SALES.1&
```

The first line accepts the month number as shown on the screen through Window 13 of Page SALES. The second line reduces the entry (in the range 1 - 12) by one (making the range 0 - 11). The third line accepts the additive amount to Window 14. The last line takes the additive amount and adds it to the amount already in the window specified by SALES.1&, or one plus WINDOWINDEX, which is the month number entered in Window 13, and stores the new total back to the window for the specified month.

MODIFYING AND EXTENDING DATAFLEX

NOTE: This section describes some unusual and powerful features of DataFlex. The vast majority of users will not have the need to make use of them. The information is included here as a matter of interest for the general user and to provide further access to the power, flexibility and utility of DataFlex for the advanced user with special requirements.

DataFlex offers the advanced configurator the unusual opportunity to actually define new command words that become an integral part of the software system--in effect, the ability to extend the command language. These operations create macros that may be used for special types of functions to satisfy highly unusual requirements, or to enable a configurator to distinctively impart his personal or company "style" to the final application product.

The commands provided with DataFlex form the basic framework for the extended "macros", and are themselves macros. Macros can be thought of as compile-time subroutines. User-defined macro commands accept arguments and textually substitute the arguments into the command model. The resultant commands are then compiled and included with the rest of the program. Using a command created through macro operations is exactly like using any other DataFlex command: the basic syntax is a command verb followed by one or more optional arguments.

Macros can be defined directly within one or more configurations, or they can be added to FMAC, the standard macro file which DataFlex refers to for the definition of all commands in a configuration at compile time. Macros defined within a configuration will operate only within that configuration when executing. The definition must appear in the source code **before** the defined macro is first used. Definitions added to FMAC on the other hand, will make the new macro available to every configuration which is compiled under it.

If a macro is defined by adding it to FMAC, the modified version of FMAC must be re-packed into the file "FLEX.CFL." FLEX.CFL is the file to which the compiler actually refers for the definition of each command it encounters in a configuration as the configuration is being compiled. The PACK program, supplied with DataFlex, is used from the operating system prompt to pack FMAC by the following command:

```
d:PACK d:FMAC
```

where D: in both places shown above represents the logical drive on which PACK.COM and the (uncompiled) FMAC files are found.

FLEX.CFL will be written to the default drive from which the above command is issued.

REPLACEMENT

The simplest form of macro operation is replacement. Any consistent string occurring in a configuration can be replaced by another string. For example, if while defining field names in FILEDEF, you establish a long, descriptive field name such as "TRANSACTION_FILE.LAST_PART_COST", you might find it tedious to type that name into a configuration a number of times. By including the following macro definition in your program, you can have the compiler substitute the long name into the compiled configuration everywhere you typed "LCOST" (or any other short string that seems descriptive to you while writing the configuration):

FORMAT:

```
      #REPLACE before_string after_string
```

EXAMPLE:

```
      #REPLACE lcost transaction_file.last_part_cost
```

Of course, the source (uncompiled) version of the configuration would always have LCOST in it, but as long as either FMAC. had the replacement in it, or the configuration had the replacement command in it **before the first** occurrence of LCOST , the compiled configuration would have TRANSACTION FILE.LAST_PART_COST substituted for each occurrence of LCOST. Especially note the initial pound sign (#) on the command line. That symbol, and certain others which will be illustrated below, comprise the distinctive syntax of macro operations.

COMMAND DEFINITION

Command definition allows you to make up new commands using sequences of existing DataFlex commands. When used in a configuration, your new command will look just like any other command in DataFlex, but it will in fact cause execution of a **sequence** of **other** DataFlex commands. Again, if the command is defined in a configuration (instead of in FMAC.), it must be defined before the command is first used in the configuration.

FORMAT:

```
        #COMMAND new_command_name {symbol list}
                (existing) command line
                additional existing command lines
                .........
        #ENDCOMMAND
```

The command may optionally be followed by a list of symbols to check whether arguments given to the command in the configuration are of the correct type(s). Symbol checking is more thoroughly explained in and after the explanation of the #CHECK command, which uses the same syntax. Do not type the {} braces; they only signify that the command component is optional.

EXAMPLE:

```
#COMMAND sign_on
        CLEARSCREEN
        SHOW
        SHOW '          EXAMPLE PROGRAM'
        SHOW ' (C) 1983 DATA ACCESS CORP'
        SHOW
#ENDCOMMAND
```

This macro defines a new command, SIGN ON. Whenever it is used in a configuration, the five indented command lines above will be executed in its place (it clears the screen and displays the two-line banner in quotation marks, surrounded by a couple of blank lines on the screen). This is useful to save typing but doesn't "expand" the language. To do that we have to be able to use arguments within our new command.

Suppose for example that we wish to be able to insert the name of the program (instead of the hard-coded legend 'EXAMPLE PROGRAM' shown above) to sign on as an argument to the SIGN ON command written into the running configuration. The syntax for an argument is an exclamation point with an integer between 1 and 9 after it. The command model will then textually substitute the first argument written after the command into the command model wherever a !1 appears in it, the second argument (if any) wherever !2 appears, and so on).

EXAMPLE:

```
#COMMAND sign_on
        CLEARSCREEN
        SHOW
        SHOW '      ' !1
        SHOW ' (C) 1983 DATA ACCESS CORP.'
        SHOW
#ENDCOMMAND
```

The "!1" used above would be replaced by the first argument following the use of SIGN ON. A use of this new command macro could be:

```
sign_on 'ACCOUNTS PAYABLE PROGRAM'
```

"ACCOUNTS PAYABLE PROGRAM" would then be substituted at the !1 position when the command was executed. Nine arguments can be replaced in this manner (i.e. !1 !2 !3....!9). The symbol "!0" is the total number of arguments that have been passed to a macro command.

A WORD OF CAUTION: Macros are not subroutines. A macro used five times in a source configuration will be expanded each time, so frequent use of a large macro can result in excessive memory consumed by the executable configuration. Generally, larger macros can be more simply installed and used within less file space if written as subroutines. The exception to this guideline is where the same macro is required across a group of several configurations.

The macro is a powerful tool. Replacements can be made anywhere within the expansion and a macro can even call other macros. The substitution for arguments is like the multiple search and replace function found in some word processors. The following utilities further enhance the utility of macros:

INCLUDE FILES

You can keep common macros and subroutines in an external file and include these at compile time with the #INCLUDE command:

#INCLUDE filename

The entire file "filename" will be included in the compilation process at the point(s) of the INCLUDE(s). One INCLUDEd file can INCLUDE yet another file. INCLUDEd files should be source (ASCII) format, not separately pre-compiled. The lines of the INCLUDEd file(s) will not be displayed during compilation, except for any #REM lines which the file may contain.

COMPILE TIME VARIABLES

Macros and configuration files can use and evaluate integer numbers and expressions at COMPILE TIME which are generally used for assigning sequential numbers to labels or variables. There are 26 such integer variables (A...Z). If an upper case letter is used, the value of the variable is incremented before use; if a lower case letter is used, the value will remain unchanged. The compile time variables are referenced with a preceding exclamation point "!". The value of the variable in text form is substituted for the "!x" in the line output.

The value of any integer variable can be set with the #SET command.

FORMAT:

#SET variable_letter$ expression

The variable can be any letter (A...Z) and should not be preceded by a "!". The "$" separates the variable letter from the expression argument. The expression can be any integer expression that can be evaluated at compile time, and it can itself contain integer variables. The expression should include parentheses.

EXAMPLE:

#SET F$ (!r-1)
SET the integer variable "F" to the value of the integer variable "r" minus one. Since "r" is lower case, it will not be automatically incremented.

CONDITIONALS:

Portions of your source lines may be selectively included or excluded from the compiled configuration by use of #IF commands. These can be used within a macro to modify its output depending on the number of arguments or their type. The IF commands set the condition and an "#ENDIF" terminates it. You may also have an "#ELSE" to change the sense of the last IF. There must be an "#ENDIF" for every "#IF", but the "#ELSE" is optional.

FORMAT

```
        #IF     expression
                command line to compile if expression TRUE
                command line to compile if expression TRUE
                etc.
        #ELSE
                command line to compile if expression FALSE
                command line to compile if expression FALSE
                etc.
        #ENDIF
```

If the expression has a value other than zero, the TRUE command lines will be included in the compiled configuration. Otherwise, the FALSE command lines will be included. For example:

```
        #IF !0>0
          SHOW 'THERE ARE !0 ARGUMENTS'
        #ELSE
          SHOW 'NO ARGUMENTS'
        #ENDIF
```

This will generate only one compiled line, the first SHOW, if there are any arguments and the second SHOW if there are not.

If we wanted to have debugging statements in our configuration that could be included or excluded based on a single statement, we could use an integer variable "d" that would be true (1) for debug mode and false (0) for runtime mode.

EXAMPLE:

```
        #SET D$ TRUE
        ...(other command lines, generating value for testvar)
        #IF !d
                SHOW 'TEST VARIABLE=' testvar
        #ENDIF
```

This example shows the debug flag initialized to TRUE, followed by configuration command lines which, among other things, generate a value for the variable TESTVAR. Where the debug flag was set TRUE, then, the value of TESTVAR will be displayed after the legend "TEST VARIABLE =". When the configuration is debugged, the #SET line can be rewritten to set the debug flag FALSE, and when the file is compiled, the lines triggering debugging statements will not be included.

The other IF commands are:

#IFDEF symbol	(TRUE if symbol has been defined.)
#IFTYPE symbol list	(TRUE if the type and class of symbol is in list)
#IFCLASS symbol list	(TRUE if the class of symbol is in list.)
#IFSAME symbol1 symbol2	(TRUE if symbol1 is the same as symbol2.)

You may check the type and class of an argument with the "#CHECK" command. #CHECK will generate an error message if the type or class of the symbol is **not** in list.

FORMAT:

#CHECK symbol list

The #CHECK command and an optional type check line on the #COMMAND line are both intended for error checking. An error will be generated if the type or class of symbol **matches** one of the type check characters in list (see symbol-checking characters below). An error will be generated if the argument is the same type or class as the symbol-checking character in list.

SYMBOL-CHECKING CHARACTERS

Character	Meaning
	Types:
S	STRING
N	NUMBER
D	DATE
I	INTEGER
E	EXPRESSION
#	INDICATOR
X	NOT AN INDICATOR
O	GROUP OPTIONS {}
B	GROUP INDICATORS []
L	LABEL
	Classes:
C	CONSTANT
F	FILE ELEMENT
W	WINDOW
V	IN MEMORY VARIABLE
G	GROUP ({options} or [indicators])
	Literal Content:
"any""characters" "in quotes"	An error will be generated if the contents of the variable(s) named in the argument do NOT match one of the strings in quotes entered in the list.
	Other:
T	ERROR IF TYPED (DEFINED)
U	ERROR IF UNTYPED
R	ARGUMENT REQUIRED
.	NO MORE ARGUMENTS
%	SPECIAL, changes constant number to constant integer.
$	SPECIAL, changes all windows to type string.

In our SIGNON example, the following would be a reasonable symbol-checking line:

EXAMPLE:

```
#COMMAND SIGNON GUR#L .
```

This would specify that SIGNON have one required argument that CAN'T be a LABEL, group, undefined or be an indicator.

The order of the type check characters has no significance. If you don't include type checking in your macros, the basic DataFlex commands will probably pick up any type errors.

The best example of using macros is the FMAC file itself.

INTERNAL TYPING OF SYMBOLS:

The following information is intended for those interested in the internal workings of the macro processor and is not required to use DataFlex.

Each symbol, whether constant, window or file, is REPLACED from its symbolic name to one which is meaningful to the DataFlex runtime system. These symbols can be identified by a vertical bar "|" in the first character of the replaced symbol. The next two characters represent the CLASS and TYPE of the symbol and the value of the symbol (if it is a constant) or its window, field and/or file numbers follows those.

EXAMPLES:

```
        |CN0    means CONSTANT, NUMBER, ZERO.
        |WS2    means WINDOW, STRING, WINDOW NUMBER TWO.
        |CI1    means CONSTANT, INTEGER, ONE
        |CS'USA'means CONSTANT, STRING, USA
        |WN3    means WINDOW, NUMERIC, WINDOW NUMBER THREE
        |FD2,1  means FIELD, DATE, FILE 2, FIELD 1
```

The following are the type and class characters used to identify symbols:

	CHARACTER	INTERPRETATION
CLASSES:		
	C	CONSTANT
	V	VARIABLE
	W	WINDOW
	F	FIELD
	G	GROUP
TYPES:		
	S	STRING
	N	NUMBER
	D	DATE
	I	INTEGER
	E	EXPRESSION
	L	LABEL
	O	GROUP OPTION
	B	GROUP BOOLEAN (INDICATOR)

If you put the command "#NOISY 99" at the top of your configuration, all of the conversion will display on the screen at compile time, together with the final output of your command lines. The option #NOISY 0 will cause only your original source code statements to display, while #NOISY 1 will cause everything to display except the intermediate code and expansions.

There is a temporary file that is generated by the compiler with a name made up of your configuration's rootname plus the extension ".IC". This is the internal DataFlex configuration code. If you use

#NOISY 99 or have an error, this file will not be deleted from your disk and can be inspected. Otherwise, the file will be deleted automatically upon completion of a successful compilation. The general format of the ".IC" file is as follows.

FORMAT:

```
        LINE_NUMBER IND1 IND2 IND3 COMMAND ARG1 ARG2
        Where INDx stands for indicator x, ARGy stands for argument y,
        and COMMAND is the internal command number.
```

Although the ".IC" file is not generally used at the configuration level it can sometimes be of interest and help in debugging difficult configurations. The list of the internal command numbers is contained on the Pascal source and library disk in the file called "COMMAND.DEF".

THIS PAGE INTENTIONALLY LEFT BLANK

RESERVED WORDS OF DATAFLEX

THE RESERVED WORDS OF DATAFLEX

The words on the next page are those used by, and built into, DataFlex. DataFlex uses these words as commands, system variables, system indicators, system procedures, options, and so on. You will recognize most of them.

They are assembled into an alphabetical list as a convenient reference to words that **must not be used** in DataFlex configurations as indicators, variable names, field names, window names, page names, file names, or any other functional string. If they are used, DataFlex is likely to mistake them for their reserved meaning and execute something in no way similar to what you intended. The words may, of course, be used as data within databases, they **may** be included in images as screen or report captions, and they **may** be used in prompting strings or any other material enclosed in quotes (single or double) within any DataFlex configuration. They may also be included within comments (material preceded by the "//" comment character) in configurations.

Unavoidably, many of the words reserved to DataFlex are words which would be very useful as variable names or for other prohibited uses. The deliberate use of small differences will effectively restore the use of these words to you while at the same time avoiding use of the actual word itself. For example, if your configuration prints the run date at the top of a report, you may not bring the date into your report by use of a variable named DATE. But it is entirely permissible to use the word **within** a longer variable name, such as RUN_DATE (note the use of underscore to simulate a space for readability, but to maintain continuity of the variable name string).

You may wish to make a photocopy of the following page to keep where it may be referred to readily while you are writing configurations. Copying this manual is expressly prohibited under the terms of your license agreement, **except** for the above-described use of the following page.

THE RESERVED WORDS OF DATAFLEX

ABORT	ENTEREND	KEY.SAVE	RANGE
ACCEPT	ENTERGROUP	KEY.SFIND	READ
ALL	ENTERMODE	KEY.UP	READLN
AND	ENTRY	KEY.USER	RECCOUNT
ANY	EQ	KEYCHECK	RECNUM
APPEND	ERASEFILE	KEYPRESS	REGISTRATION
AS	ERR	KEYPROC	RELATE
ASCII	ERRLINE	LASTERR	RENAME
ATTACH	ERROR	LASTIF	REPEAT
AUTOFIND	FALSE	LE	REPORT
AUTOPAGE	FIELDINDEX	LEFT	REPORTEND
BACKFIELD	FILELIST	LENGTH	REQUIRED
BEGIN	FILENUMBER	LINECOUNT	REREAD
BLANKFORM	FILL	LOCK	RETAIN
BREAK	FIND	LOOP	RETAINALL
BREAKINIT	FINDERR	LT	RETURN
BREAKPOINT	FINDREQ	MATCH	RIGHT
BY	FLEXKEY	MEMAVAIL	RPT.anything
CALC	FLOAT$	MID	RUNPROGRAM
CALCULATE	FOR	MOVE	SAVE
CAPSLOCK	FORCEPUT	MOVEINT	SAVERECORD
CHAIN	FORMAT	MOVENUM	SCREENEND
CHARACTER	FORMFEED	MOVESTR	SECTION
CHECK	FOUND	MULTIUSER	SELECT
CLEAR	FROM	NAME	SEQEOF
CLEARFORM	GE	NE	SEQEOL
CLEARSCREEN	GOSUB	NEWPAGE	SETCHANGE
CLEARWARNING	GOTO	NEXT	SHOW
CLEARXY	GOTOXY	NOENTER	SHOWLN
CLOSE_INPUT	GROUP	NOISY	SKIPFOUND
CLOSE_OUTPUT	GT	NOPUT	STATUS
CMDLINE	HEADER	NOT	STRING
COPYFILE	HELP	NUMBER	STRLEN
DATE	IF	NUMPAGE	STRMARK
DEBUG	IFCHANGE	NUMWINDOW	SUBHEADER1...9
DELETE	IFNOT	ON	SUBTOTAL
DESPOOL	IN	OPEN	SUBTOTAL1...9
DIRECT_INPUT	INCREMENT	OR	SUPPRESS
DIRECT_OUTPUT	INDEX.BATCH	OUTCLOSE	SYSDATE
DIRECTORY	INDEX.1...10	OUTFILE	SYSTEM
DISPLAY	INDICATE	OUTPUT	TERMCHAR
DISPLAYONLY	INDICATOR	PAD	THRU
END	INKEY	PAGE	TO
END$OF$REPORT	INPUT	PAGE.LINES	TOTAL
END.OF.REPORT	INTEGER	PAGEBREAK	TRIM
ENDGROUP	KEY.CLEAR	PAGECHECK	TRUE
ENT$PERMISSIVE	KEY.DELETE	PAGECOUNT	UNLOCK
ENTAGAIN	KEY.DOWN	PAGEEND	UNTIL
ENTDISPLAY	KEY.ESCAPE	PAGEFEED	UPPERCASE
ENTER	KEY.FIELD	PAGE1...255	WHILE
ENTER.CLEAR	KEY.FIND	PATHNAME	WINDOWINDEX
ENTER.DELETE	KEY.HELP	POINTS	WRITE
ENTER.EDIT	KEY.NEXT	POS	WRITELN
ENTER.EXIT	KEY.PREVIOUS	PRINT	ZEROFILE
ENTER.SAVE	KEY.RETURN	QUIET	

FILES, ARGUMENTS AND COMMANDS

FILE NAME AND EXTENSION TYPES

Each DataFlex database is actually composed of a "family" of operating system files which function with each other for the DataFlex user as a single unified whole. The data file "rootname" (specified by the user in the FILEDEF routine) is common to all the members of a family of data files, while differing extensions denote various standard functions for each of the different files.

The following are the standard extensions and file names used by DataFlex. The "*" indicates a file required for runtime. The "#" indicates a file required only at compile time. SOLID CAPITAL letters denote explicit content (usually extensions). Lower case letters are descriptive of a filename which may be virtually any unique string of characters recognized by the CP/M operating system as a file name element.

FILE NAME	TYPE	DESCRIPTION
rootname.DAT	*RAND	DataFlex data file. Contains file definition and data (one per database)
rootname.K??	*RAND	DataFlex index file (one for each index)
rootname.DEF	SEQ	File definition listing
rootname.TAG	#SEQ	Field tag names file
FILELIST.CFG	*RAND	List of active file numbers and screen control codes for configured terminal
TERMLIST.CFG	RAND	Available CRT configurations pre-coded
program.COM	PROG	Executable program (CP/M and MS-DOS systems)
program.CMD	PROG	Executable program (CP/M-86 systems)
module.ERL	SEQ	Unlinked Pascal MT+ library module
program.OVF	PROG	Program overlay file
config.FLX	*FLEX	Executable DataFlex configuration
config.FD	#SEQ	File definition info for compiler
config.IC	SEQ	Intermediate DataFlex code file.
config.RPT	SEQ	Source code for sample reports
config.FRM	SEQ	Source code for sample screens
SHOWDEMO	SEQ	Source code for COMDEX show demo
FLEX.CFL	#RAND	Packed macro command file
FMAC	SEQ	Unpacked macro command file
PACK.COM/CMD	PROG	Program to pack FMAC (command=PACK FMAC)
ENTER	SEQ	List of the ENTER macro
REPORT	SEQ	List of the REPORT macro
FLEX.COM/CMD	*PROG	DataFlex runtime program
RUN.OVF	*OVF	DataFlex runtime overlay file
COMP.COM/CMD	#PROG	DataFlex compile program
COMP.OVF	#OVF	DataFlex compile overlay file
READ.FLX	FLEX	Program generator to read sequential files

NOTES: .OVF and .FD files must reside on the logged-in disk drive.
16-bit versions do not have, or use, COMP.OVF
In 16-bit versions, RUN.OVF is replace by RUN.001 thru .004

PREDEFINED ARGUMENTS

The folllowing are the predefined indicators and system variables discussed in sections of the manual to which they pertain:

INDICATORS:

ENT$PERMISSIVE	Disables resetting of FIND and AUTOFIND
ERR	Error indicator
FINDERR	Result of last FIND (opposite of FOUND)
FOUND	Result of last FIND
KEY.CLEAR	Clear screen Flex-Key
KEY.DELETE	Delete record Flex-Key
KEY.DOWN	Down arrow Flex-Key
KEY.ESCAPE	Escape, end function Flex-Key
KEY.FIELD	Back Field, window Flex-Key
KEY.FIND	Find Flex-Key
KEY.HELP	Help Key Flex-Key
KEY.NEXT	Next record Flex-Key
KEY.PREVIOUS	Previous record Flex-Key
KEY.RETURN	Return/enter Flex-Key
KEY.SAVE	Save record Flex-Key
KEY.SFIND	Super Find Flex-Key
KEY.UP	Up arrow Flex-Key
KEY.USER	User-defined Flex-Key
KEYPRESS	Set by KEYCHECK, if key press
LASTIF	Result of most recent IF test
MULTIUSER	Set by MULTIUSER system option
PAGEBREAK	If new page, set by PAGECHECK
SEQEOF	Sequential end of file
SEQEOL	Sequential end of line
SELECT	Triggers selection in REPORT macro
STATUS	Shows whether file named is ACTIVE
SUBTOTAL1..9	Breakpoint indicators

SYSTEM VARIABLES:

ERRLINE	Configuration line number of last error
FILENUMBER	Number of last DataFlex file for FILELIST
FIELDINDEX	Indexed addressing for fields
FLEXKEY	Number of last Flex-Key used
LASTERR	Number of most recent error triggered
LINECOUNT	Output line counter
MEMAVAIL	Amount of memory of available
NUMPAGE	Number of pages
NUMWINDOW	Number of windows
PAGECOUNT	Output page counter (set by PAGECHECK)
PAGEEND	Printable lines per page
PAGEFEED	Action to take on page feed
RECCOUNT	Running total of records selected in REPORT
STRLEN	String length
STRMARK	String position marker
WINDOWINDEX	Indexed addressing for windows

WINDOW FORMAT AND ENTRY OPTIONS

The brackets {} shown below are part of the command syntax, and should be included with the desired option(s) when written into configurations.

WINDOW FORMAT OPTIONS:

	OPTION	SYNTAX	EFFECT
	DECIMAL	{POINTS=0}	Numeric decimal points
*	FILL	{FILL="*"}	Fill character
*	FLOAT$	{FLOAT$}	Floating "$" sign
	RANGE	{RANGE=1,10}	Numeric range checking
*	SIGN ON RIGHT	{SIGNRT}	Minus sign on right
*	SUPPRESS	{SUPPRESS}	Space suppression
	UPPERCASE	{CAPSLOCK}	Force upper case
	VALIDITY	{CHECK="Y|N"}	Valid response checking

ENTRY OPTIONS

AUTOMATIC FIND	{AUTOFIND}	FIND record automatically
DECIMAL POINTS	{POINTS=0}	Numeric decimal points
DISPLAY ONLY	{DISPLAYONLY}	Cursor always skip window
FIND REQUIRED	{FINDREQ}	Successful FIND required
FORCE PUT	{FORCEPUT}	Update buffer unconditionally
NO PUT	{NOPUT}	Do NOT put to the database
NO ENTER	{NOENTER}	No data entry to window
RANGE CHECKING	{RANGE=1,10}	Numeric range checking
REQUIRED	{REQUIRED}	Required entry.
RETAIN	{RETAIN}	Retain window, clearable
RETAIN ALWAYS	{RETAINALL}	Retain window, not clearable
SKIP IF FOUND	{SKIPFOUND}	Skip entry if record found
UPPER CASE	{CAPSLOCK}	Force upper case
VALIDITY	{CHECK="YN"}	Valid response checking

*Option may be used by PRINT command only.

COMMAND SYNTAX LIST

<Darts> indicate required substitutive elements (do **not** type darts).
{Brackets} indicate optional substitutive elements (do **not** type brackets except where they are shown below in double {{ and }}).
[Braces] enclose GROUPed indicators, and **are** to be typed.
"Arg" signifies an argument which may be a database element, window, or in-memory variable.
Three periods (...) indicate a non-specific quantity of the elements which appear on either side of the periods.

```
CONSOLE I/O GROUP
        CLEARSCREEN
        CLEARXY <line> <column>
        GOTOXY <line> <column>
        INKEY <arg>
        INPUT {'prompt'} <arg>
        KEYCHECK {command}
        SHOW <arg> {arg ... arg}
        SHOWLN <arg> {arg ... arg}

CONTROL GROUP
        ABORT
        CLEARWARNING
        CHAIN "<configuration> {string ... string}"
        DEBUG
        ERROR <number>
        GOTO <label>
        GOSUB <label>
        ON <integer_arg> GOSUB <label> {label ... label}
        ON <integer_arg> GOTO <label> {label ... label}
        RETURN {label}

DATABASE GROUP
        ATTACH <filename>
        CLEAR <filename> {filename ... filename}
        DELETE <filename> {filename ... filename}
        FIND <mode> <element>      MODES=LT,LE,EQ,GE,GT
        OPEN <file name> {INDEX.n}
        RELATE <filename>
        SAVE <filename> {filename ... filename}
        SAVERECORD <filename> {filename ... filename}
        ZEROFILE <filename>

DEFINITION GROUP
        DATE <date> {date ... date}
        INDICATOR <indicator> {indicator ... indicator}
        INTEGER <integer> {integer ... integer}
        NUMBER <number> {number ... number}
        STRING <string> {length}
```

```
ELEMENT PROCESSING GROUP
        CALC{ULATE} <expression> to <arg>
        INCREMENT <integer>
        MOVE <arg> TO <arg>
        MOVEINT <arg> TO <integer>
        MOVENUM <arg> TO <number>
        MOVESTR <arg> TO <string>

ENTER MACRO GROUP
        ENDGROUP
        ENTDISPLAY
        ENTER <filename> {filename ... filename}
        ENTEREND
        ENTERGROUP
        ENTRY <file.field> {window {options}}

FORMS GROUP
        ACCEPT <window> {TO <arg>}
        AUTOPAGE <pagename>
        BLANKFORM <pagename>
        BLANKFORM <window> {THRU <window>}
        CLEARFORM <pagename>
        CLEARFORM <window> {THRU <window>}
        DISPLAY <arg> {TO <window>}
        FORMAT <window> {{options}}
        NAME <pagename.#> <windowname> {windowname ... windowname}
        OUTCLOSE
        OUTFILE <device>
        OUTPUT <pagename>
        PAGE <pagename>
        PRINT <arg> {TO <window>}

INDICATE GROUP
        IF <arg> <mode> <arg> <command>
        INDICATE {NOT} <indicator> AS <arg> <mode> <arg>
                MODES:  LT, LE, EQ, GE, GT, NE, IN, MATCH
        INDICATE {NOT} <indicator> GROUP ANY/ALL [indicator indicator]
                {AND/OR ANY/ALL [indicator indicator {indicator}]}
        INDICATE {NOT} <indicator> STATUS <filename>
        INDICATE <indicator> TRUE
        INDICATE <indicator> FALSE

KEY GROUP
        BACKFIELD
        ENTAGAIN
        ENTERMODE
        KEYPROC <flexkey>

MULTI-USER GROUP (compile, but do not execute under single-user)
        DESPOOL
        IFCHANGE <arg> <command>
        LOCK
        REREAD {filename ... filename}
        UNLOCK
```

```
REPORT MACRO GROUP
        FORMFEED
        OUTPUT <section>
                SECTIONS:  HEADER, SUBHEADER#, BODY, SUBTOTAL, TOTAL
        PAGECHECK <lines>
        REPORT <file> {BY <index.n or RECNUM>} {BREAK file.field
                file.field ... file.field}
        REPORTEND
        SECTION <section>

SEQUENTIAL I/O GROUP
        CLOSE INPUT
        CLOSE_OUTPUT
        DIRECT_INPUT <device>
        DIRECT_OUTPUT <device>
        READ <arg> {arg ... arg}
        READLN <arg> {arg ... arg}
        WRITE <arg> {arg ... arg}
        WRITELN <arg> {arg ... arg}

STRING GROUP
        APPEND <destination string> <string> {string ... string}
        ASCII <string> TO <number>
        CHARACTER <number> TO <string>
        CMDLINE <string_variable>
        LEFT <string_variable> TO <string_variable> <length>
        LENGTH <string_variable> TO <number>
        MID <string_variable> TO <string_variable> <length> <start>
        PAD <string_variable> TO <string_variable> <length>
        POS <string_variable> IN <string_variable> TO <number>
        RIGHT <string_variable> TO <string_variable> <length>
        TRIM <string_variable> TO <string_variable>
        UPPERCASE <string_variable>

STRUCTURED CONTROL GROUP
        BEGIN - END
        FOR <integer> FROM <arg> TO <arg> - LOOP
        REPEAT - UNTIL <arg> <mode> <arg> - LOOP
        WHILE <arg> <mode> <arg> - END
                MODES:  LT, LE, EQ, GE, GT

SYSTEM GROUP
        COPYFILE <sourcefile(s)> TO <destfile(s)>
        DIRECTORY <specifications>
        ERASEFILE <filename>
        FILELIST <filenumber> TO <display name> {filename}
        REGISTRATION
        RENAME <filename1> TO <filename2>
        RUNPROGRAM <prog_name.com> {program_arg}
        SYSDATE <date> <number> {number}
        SYSTEM
```

SPECIFICATIONS

Provided by DataFlex:	File Maintenance (on line, interactive) File Update (on line, interactive) Data Entry (on line, interactive) Report Generation Query (on line, interactive) Command language Data Base Management Pascal Library (optional) Program-independent menu system
Microprocessor Environment:	8080, Z-80, 8085 etc. 8086, 8088, 80186, 80286 etc.
Supported Operating Systems:	CP/M, CP/M-86, MP/M-86, MS-DOS, PC-DOS, TurboDOS, Novell Sharenet, PC-Net, 3COM, DMS Hi-Net, Molecular N-Star, Televideo MmmOST, Action DPC/OS, Omninet, Network, IBM "PC" with Corvus
Requirements:	Transient Program Area: (8 bit): 52k (16 bit): 100k CRT with addressable cursor 600k of disk storage
Purpose:	Applications Development
Structure:	Extended Relational DBMS With Data Independent Utilities and command language.
Host Language:	Pascal/MT+ (c) Digital Research) (Not required for execution or configuration.) NOTE: Many DataFlex utilities are themselves actually DataFlex configurations.
Maximum DBMS Files:	125
Maximum data elements per file:	255
Maximum indexes per file:	(8 bit): 4 (16 bit): 9 Plus one "ad hoc" index.

Maximum elements per index:	(8 bit): 4 (16 bit): 6
Maximum File size:	8 Megabytes
Maximum records per file:	65,536
Maximum record size:	4,000 bytes to *MEML
Indexing:	B+ Multi-level ISAM
Data File type	Packed, fixed-length random access
Numeric Type:	Packed BCD fixed point.
Numeric precision:	4 places after decimal point
Numeric range:	+ or - 99,999,999,999,999.9999
Maximum data files open simultaneously:	(*MEML) only (8 bit): at least 5 (16 bit): at least 10 (plus associated index files)
Maximum configuration lines:	32,000 (*MEML)
Maximum line length:	255 characters
Maximum argument size:	80 characters
Maximum number of windows:	32,000 (*MEML)
Maximum pages per configuration:	255 (disk or memory resident)
Maximum number of indicators:	89 (plus 38 pre-defined)
Maximum number of variables:	32,000 (*MEML)
Maximum INTEGER variables:	40
Integer range:	+ or - 32,767
Flex-Keys:	18, terminal-independent
Sub-category breakpoint levels:	9
Sequential files:	Line- or comma-delimited Device-independent.
Command file characteristics:	Protected source, semi-compiled.

Types of arguments:	Text strings Numeric (bcd) Integer Date (extended Julian)
Types of commands:	Data Movement & conversion Indicators (conditionals) Calculation Control, unstructured Control, Structured IMAGE forms Data Entry Reporting Data Base Manipulation String Sequential I/O Console I/O (terminal-independent) Macro pre-processor
Utilities:	Data Base Definition Data Base recovery Terminal & system installation Text editor Program generation Query Configurable MENU Configuration compiler

*MEML - Memory limited, affected primarily by the available memory (TPA) in the computer system and the memory consumed by the rest of the configuration.

THIS PAGE INTENTIONALLY LEFT BLANK

COMPILER ERROR MESSAGES

TYPES OF ERRORS

COMPILE time and RUN time: It is important to distinguish those actions and errors which occur when a program is compiled from those which occur when a program is run. The DataFlex compiler takes your English command source code and translates it into a "compiled" form that is more efficient to execute. This compiled form is not machine language, but an internal form unique to DataFlex.

When this translation (compilation) takes place, the compiler may run into a command line which it doesn't know how to translate; this is called a compile time error. Most errors in the use of commands, as well as typographical errors, are caught at compile time, which guarantees that your program is syntactically correct when it runs. Other errors may be caused by the action of the operator, the data which is acted upon by the program, or faulty logic by the programmer. These errors are usually flagged when the compiled program is actually run (runtime). For this reason, compile time errors and runtime errors are separated into separate groups. Be sure to look at the correct table when trying to diagnose an error!

CONFIGURATION SYNTAX ERRORS

If you misspell or misuse a DataFlex reserved word in a command line, you will get a syntax error when you compile the configuration. These errors are the simplest to find and the easiest to correct. Simply edit the configuration, make corrections on the indicated line(s), and recompile. You may need to refer to the manual to determine the required syntax for the command you are trying to use. Pay particular attention to the arguments of the command and what type of argument is reasonable and legal to put in a particular position (e.g., trying to move a new value into a constant is both illegal and unreasonable). Most syntax errors are caused by simple misspellings or typographical errors. Very few syntax or usage errors will occur at runtime. Those which do usually arise from use of window- and field-indexing, which can change a variable's type at runtime.

Another type of compile time error is the use of duplicate symbols. If you use the same name for two variables or labels, you will get an error. Each label and symbol must be unique! (DataFlex won't detect duplicate use of a variable if both uses are syntactically consistent with each other, so this is not a total failsafe.) You must also avoid the use of DataFlex reserved words. These reserved words (listed in the section on "Reserved Words of DataFlex") canNOT be used as symbol names. It IS legal to use symbol substitution for command names. This will not generate an error, but great care must be exercised and this practice is not recommended. If you use a command name as a symbol, you will get an error when you then try to use that command. In summary, when naming your files, pages, variables and labels, make sure to use unique names which are not reserved words.

The DataFlex compiler program (COMP.COM) completes four processes in two passes through the source file. There is a class of errors for each process:

```
        1) Initialization -----.
        2) Forms               |-------> PASS ONE
        3) Macro          -----'
        4) Compile--------------------> PASS TWO
```

INITIALIZATION ERRORS:

SOURCE FILE NOT FOUND
Check disk directory for correct file name.

FORMS ERRORS:

IMAGE NOT FOUND
Check that first line of file begins with "/". If no image desired begin with "/*".

MACRO ERRORS:

MEMORY: 'amount_memory_available'
Warning updated every time a new symbol is encountered if remaining memory is less than one kilobyte. Displays amount of memory remaining available. If "Out of Memory" error is declared, see compiler option "M" for possible fix.

150 MACRO BUFFER SIZE EXCEEDED
Macros as defined are too large to fit in available buffer space. Usually the result of a syntax error. Otherwise, compiler option "M" may permit compilation with larger macro expansion buffer.

151 INCLUDE FILE NOT FOUND
Check device directory to ensure that the specified file exists.

152 UNEQUAL PARENTHESES IN EXPRESSION
Open and close parentheses not matched.

155 INDICATOR NOT RESOLVED: 'symbol_name'
Indicator used has not been defined. Often caused by misspelling.

155 TOO MANY INDICATORS
Maximum of 3 indicators per line exceeded.

156 NO ACTIVE CONDITIONALS
A #ELSE appears in a macro without a #IF being open.

157 FORWARD REFERENCE: 'symbol name' NOT RESOLVED
Label (GOTO or GOSUB) not found. Often caused by spelling inconsistencies. If the error is not apparent in the source file, check whether one of the commands may be using an internal GOTO or GOSUB.

159 TOO MANY CONTROL BLOCKS
The maximum number of control blocks which may be open is 20. This error is triggered when there are more than 20 open at one time. Control blocks are created by IF statements and structured control commands, among other things.

160 TERMINATED CONT'L BLOCK W/O BEGIN
A control block terminator appears without a preceding block initializer. Examples are END (requires BEGIN or WHILE), UNTIL or LOOP (require REPEAT), etc.

162 TYPE CHECK ERROR IN ARGUMENT: ARGUMENT REQUIRED
The command was issued with fewer than the minimum required number of arguments.

162 TYPE CHECK ERROR IN ARGUMENT: CONSTANT NOT ALLOWED

162 TYPE CHECK ERROR IN ARGUMENT: DATE NOT ALLOWED

162 TYPE CHECK ERROR IN ARGUMENT: EXPRESSION NOT ALLOWED

162 TYPE CHECK ERROR IN ARGUMENT: FILE ELEMENT NOT ALLOWED

162 TYPE CHECK ERROR IN ARGUMENT: GP INDICATOR "[]" NOT ALLOWED

162 TYPE CHECK ERROR IN ARGUMENT: GROUP NOT ALLOWED

162 TYPE CHECK ERROR IN ARGUMENT: INDICATOR NOT ALLOWED

162 TYPE CHECK ERROR IN ARGUMENT: INTEGER NOT ALLOWED

162 TYPE CHECK ERROR IN ARGUMENT: INVALID LITERAL

162 TYPE CHECK ERROR IN ARGUMENT: LABEL NOT ALLOWED

162 TYPE CHECK ERROR IN ARGUMENT: MUST BE AN INDICATOR

162 TYPE CHECK ERROR IN ARGUMENT: MUST BE DEFINED

162 TYPE CHECK ERROR IN ARGUMENT: NUMBER NOT ALLOWED

162 TYPE CHECK ERROR IN ARGUMENT: OPTION "{}" NOT ALLOWED

162 TYPE CHECK ERROR IN ARGUMENT: SHOULD NOT BE DEFINED

162 TYPE CHECK ERROR IN ARGUMENT: STRING NOT ALLOWED
A string, or string variable, was used in a command which does not deal with strings (uses numerics, etc.).

162 TYPE CHECK ERROR IN ARGUMENT: TOO MANY ARGUMENTS
Multiple arguments were used with a command which allows only one argument, or multiple arguments were used with a command which can handle only a certain number (see explanations of commands in other sections of this manual).

162 TYPE CHECK ERROR IN ARGUMENT: VARIABLE NOT ALLOWED
A variable name was used in a command which cannot deal with variables.

162 TYPE CHECK ERROR IN ARGUMENT: WINDOW NOT ALLOWED
A window was used in a command which cannot deal with windows.

164 COMMAND NOT FOUND: 'symbol name'
A command in the source file is not present in FLEX.CFL Misspellings frequently trigger this error. Not rePACKing FMAC after modifying it for new macros can trigger this error, as can use of a reserved word as a variable or other improper use. This error occurs if FLEX.CFL is not on the default drive.

170 CANNOT REPLACE TO ITSELF
Logic of replacement is not valid. Often caused by misuse of reserved word(s) or syntax error(s).

COMPILE TIME ERRORS

ICODE FILE NOT FOUND
Intermediate code file from first pass of compilter not found. Usually a failure of hardware or operating system. A disk full condition is a frequent cause of this error.

101 NUMBER EXPECTED
A numeric expected but not found. ICODE error in the command sequence.

102 LINE NUMBER OUT OF SEQUENCE
Internal code inconsistency. Error in a user-defined macro.

103 ILLEGAL VARIABLE TYPE
Compiler type check of the ICODE.

104 DUPLICATE VARIABLE DEFINED
Same variable defined twice as different types. Coding error.

105 TYPE/CLASS NOT FOUND
Internal compiler conflict caused by a coding error.

106 ERROR IN WRITING .FLX FILE
Operating system error (Disk or Directory Full).

107 ERROR IN CLOSING .FLX FILE
Operating system error (Disk or Directory Full).

108 OPEN PARENTHESIS EXPECTED
Start of an expression not found.

109 UNEQUAL PARENTHESES
End of an expression not found.

111 INVALID ARGUMENT
Internal compiler conflict. Unresolved symbol encountered.

112 INVALID ENTRY OR FORM COMMAND
Check options "{}" for validity and spelling.

122 LINE TOO LONG
The maximum line length allowed is 255 characters. This error is triggered by any line longer than that. Check for omission of carriage return/line feeds from the source file.

THIS PAGE INTENTIONALLY LEFT BLANK

RUNTIME ERRORS AND DIAGNOSIS

TYPES OF ERRORS

Runtime errors are those which occur when you are actually running a configuration (as opposed to compiling it). Compiler errors are discussed in the preceding section of this manual. Runtime errors can be the result of a fault in a file definition, the specifications made in the installation of DataFlex, configuration of your operating system, or operator actions, but most runtime errors result from a condition which has to be corrected in the configuration. When this is the case, corrective action usually includes the same steps required to correct compiler errors: edit the configuration, make corrections, and recompile the configuration. Other steps (discussed further at several points below in this section) may be required in addition to, or instead of, these steps.

RUNTIME ERROR CLASSIFICATIONS:

Runtime errors are discussed in this section in the following groups. If you are dealing with a particular error, find it in the numerical list below and review the discussions under the classification(s) to which it is assigned. A given error may be discussed under more than one classification where the error may be triggered by more than one cause.

OP:	OPERATOR ERRORS
MH:	MEDIA AND HARDWARE ERRORS
L/D:	LIMITATION, DISK
L/M:	LIMITATION, MEMORY
L/C:	LIMITATION, CONFIGURATION
CE:	CONFIGURATION ERROR
SU:	SET UP ERROR
MU:	MULTIUSER ERROR

ERROR #	DESCRIPTION	CLASSIFICATION
OPERATING SYSTEM ERRORS:		
1	READING UNWRITTEN DATA (SYSTEM)	MH
2	OPERATING SYSTEM ERROR	MH
3	CANNOT CLOSE CURRENT EXTENT	MH
4	SEEK TO UNWRITTEN EXTENT (SYSTEM)	MH
5	DIRECTORY OVERFLOW (DISK FULL)	L/D
6	SEEK PAST END OF DISK (DISK FULL)	L/D

ERROR #	DESCRIPTION	CLASSIFICATION
DATAFLEX ERRORS:		
10	+++ OUT OF MEMORY +++	L/M,CE
11	NUMBER TOO LARGE FOR FIELD ALLOCATION	OP, CE
12	WINDOW NUMBER OUT OF RANGE	CE
13	AN ENTRY IS REQUIRED ON THIS WINDOW	OP
14	PLEASE ENTER A NUMBER	OP
15	INVALID ENTRY FOR THIS WINDOW	OP
16	PLEASE ENTER A VALID DATE (MM/DD/YY)	OP
17	NUMERIC ENTRY IS OUT OF RANGE	OP
18	CAN'T OPEN SYSTEM COMMUNICATIONS FILE	CE
19	MULTIUSER TIME OUT (BUSY TOO LONG)	CE
20	READ ERROR ON INDEX FILE, REINDEX FILE	MH
21	WRITE ERROR ON INDEX FILE	MH, L/D
22	INDEX FILE DAMAGED, REINDEX FILE	MH
23	INDEX FILE FULL, EXCEEDS DEFINED SIZE	L/C
25	RECORD NOT FOUND	OP
26	CAN'T CLOSE INDEX FILE	MH, MU
28	DUPLICATE RECORDS NOT ALLOWED IN FILE	OP, CE
30	CAN'T READ CONFIGURATION FILE	CE, MH
31	CONFIGURATION FILE NOT FOUND	OP, CE
32	CAN'T OPEN OUTPUT FILE	CE, MH, L/D
41	FIND PAST BEGINNING OF FILE	OP
42	FIND PAST END OF FILE	OP
43	CAN'T OPEN INDEX FILE	SU, MU
49	DEMO VERSION LIMITATION EXCEEDED	L/D,BUY DATAFLEX
51	BAD FORMAT IN EXPRESSION (OPERAND)	CE
52	BAD FORMAT OF EXPRESSION (OPERATOR)	CE
71	NO RECORD IN MEMORY TO DELETE	OP, CE
72	FILE NOT OPEN	SU, MU
73	FIELD NUMBER OUT OF RANGE	CE
74	CAN'T OPEN "FILELIST.CFG"	SU, MU
75	CAN'T OPEN DATA FILE (.DAT)	SU, MU
76	FIELD NUMBER OUT OF RANGE	CE
77	FIELD NUMBER OUT OF RANGE	CE
78	CAN'T UPDATE RECORD ZERO OF DATA FILE	MH, MU
79	FIELD NOT INDEXED, CAN'T FIND BY THIS	OP, CE
80	CAN'T CLOSE DATA FILE	MH, MU
81	RECORD NUMBER OUT OF RANGE	MH, MU
82	EDITED RECORD NOT SAVED	CE
83	TYPE CHECK ERROR, ASCII USED AS NUMBER	CE
85	ISAM NOT LINKED TO PROGRAM	PS
86	RELATED FIELDS ARE NOT THE SAME LENGTH	CE
87	NO SUPERFIND PATH TO THIS RECORD	OP
88	INVALID FILE NAME	OP
90	PLEASE ENTER A VALID RECORD ID	OP
91	ATTEMPT TO PUT INTO INTEGER CONSTANT	MACRO COMMAND
92	CONFIGURATION FILE NOT FOUND	OP, SU, CE
96	CAN'T OPEN CONFIGURATION/PROGRAM FILE	OP, SU, CE
97	MULTIPLE GOSUB'S WITHOUT RETURN	CE
100	OPERATOR ERROR	OP

OP: OPERATOR ERRORS

The most common error is a simple mistake by the operator. DataFlex handles many operator errors directly and gives you the capacity for providing additional checks where required. Most of these errors, unlike the other runtime errors, do not call for any corrections in configurations or system setup. The most correction that is likely to be needed is operator instruction in those cases where the error message itself is not sufficient for the purpose. The following are errors that can be generated by an operator in a properly written configuration.

ERROR 11: NUMBER TOO LARGE FOR FIELD ALLOCATION
Number is too large for a data field

ERROR 13: AN ENTRY IS REQUIRED ON THIS WINDOW
Operator attempted to return past a "REQUIRED" window

ERROR 14: PLEASE ENTER A NUMBER
Entry of letters ("O", lower case "L", etc.) in a numeric-typed window

ERROR 15: INVALID ENTRY FOR THIS WINDOW
Entry does not conform to CHECK or format specification made in the configuration. Some means for determining what is acceptable must be provided for the operator (Help screen, prompt, custom error message)

ERROR 16: PLEASE ENTER A VALID DATE (MM/DD/YY)
Improper date format or value

ERROR 17: NUMERIC ENTRY IS OUT OF RANGE
Entry does not conform to "RANGE" in entry option made in the configuration. Some means for determining what is acceptable must be provided for the operator (Help screen, prompt, custom error message)

ERROR 28: DUPLICATE RECORDS NOT ALLOWED IN FILE
An attempt was made to enter two records with the same key

ERROR 41: FIND PAST BEGINNING OF FILE
Previous record find attempted at beginning of file

ERROR 42: FIND PAST END OF FILE
Next record find attempted at end of file

ERROR 71: NO RECORD IN MEMORY TO DELETE
Delete key pressed with no record found

ERROR 87: NO SUPERFIND PATH TO THIS RECORD
Can't superfind from this window; main file does not relate to this field

ERROR 88: INVALID FILE NAME
A badly formatted or too long file name has been entered

ERROR 90: PLEASE ENTER A VALID RECORD ID
No record is found with a "FINDREQ"; press FIND

ERROR 92: CONFIGURATION FILE NOT FOUND
Operator entered a configuration file name that is not found on the indicated disk (also see Error 96).

MH: MEDIA AND HARDWARE ERRORS

If there is a problem with your equipment, disk media or operating system, it will usually show up in the filing system. If a program which was working properly suddenly stops working, it is usually (but not absolutely always) some sort of hardware or media problem. A problem which does not happen consistently under exactly the same conditions is almost always a hardware problem. If you are having this type of problem, you should run all of your system diagnostics several times. You should also test other software which uses similar resources. After any hardware malfunction, the data recovery utilities REINDEX and FREL should be used (see the section on "Utilities").

Media and hardware errors refer to problems which result in physical corruption of the data on your disk, not operator or program error. There are two primary causes for these errors: disk media defects and power failure. There is a good statistical probability that eventually you will have a media failure. It is equally true that you will have power failures. DataFlex keeps all of your data current on disk in case of power failure, but different equipment and operating systems have different capacities for coping with this problem. Some poorly designed equipment can destroy an entire disk if the power goes out at the wrong time, while others handle it with almost no effect. Probably the worst case is when a system makes small, almost undetectable errors on the disk after a power failure. Some power "flickers" or "spikes" can cause data errors which are unnoticed at the time they occur. After a power failure, flicker or spike, or a media error, you should use the data recovery utilities FREL and REINDEX to ascertain the extent of data corruption and re-establish data integrity. It is most important to MAINTAIN RECENT BACKUPs to guard against these data killers. If you rely heavily on your system, you should consider an uninterruptible power supply to prevent power failure problems.

DATA RECOVERY TECHNIQUES: WHY WOULD YOU NEED TO RECOVER DATA?

Due to power problems, media failure and sometimes configuration errors, it may become necessary to restore the integrity of your data. This recovery may be necessary on several levels. If you have a corrupted disk directory, you will need to recover all of the files on a disk (primary data recovery). If only one file is corrupted, there are specific methods for recovering one file (secondary data recovery).

Before you start any kind of data recovery you should consider whether it is worthwhile. Assuming you have made proper backups, it may be simpler to restore your back ups and bring those files up to date by manually re-inputting data!

PRIMARY DATA RECOVERY: CORRUPTED DIRECTORY

If the directory on your disk is corrupted (see media and data errors), you will need to restore the integrity of the disk before you start on the individual data files. You should first make ANOTHER backup of your entire disk drive and then reformat the disk according to the operating system instructions. Next, verify the disk drive to make sure there are no media errors. Finally, restore the fields from your backup. You should examine all data files for correctness and proceed with secondary file recovery on ALL files.

SECONDARY DATA RECOVERY: CORRUPTED FILE

If the integrity of any file is suspect, you should use both the FREL and the REINDEX utilities on each file. The FREL utility will restore the internal list of deleted records in the .DAT file and the REINDEX utility will recreate the indexes. If there are any bad or duplicate records in the file, they will be removed. After this procedure, it's a good idea to use QUERY to look at the data in the files.

If the recovery programs cannot recover the badly damaged data, it is usually better to restore the backup files than to try further recovery. Let us emphasize the importance of good back ups!

Common errors associated with media and hardware failures are:

SYSTEM STOPS OR "LOCKS UP" at random times
    Most probably a power flicker or spike! (Also check memory available)

BDOS Error: Bad Sector; I/O Error; Read/Write Error
    These errors are returned directly from the operating system, not from DataFlex. The exact wording may vary from system to system. This type of error is caused by an unrecoverable hardware or media error. The first thing to do is to reformat and restore your data from backup. If this happens frequently, your hardware should be examined. If you use floppy disks, you should consider another brand.

ERRORS 1, 2, 3, 4
    These errors generally indicate problems with the operating system's directory structure of the disk drive. Power failure is a common cause of directory corruption. If you have had other media errors, they can show up later as directory problems. Generally the disk should be reformatted and the data recovery techniques used. (See also Disk Limitations)

ERRORS 20, 21, 22, 26
    Errors 20, 22 and 26 are caused by a corrupted index file, generally caused by the same types of errors as above. Error 21 can also be caused by a disk full condition. The data recovery utilities should be sufficient. If this happens repeatedly, there may be a subtle problem with the operating system or equipment. If you are on a multiuser system, make sure that you follow the multiuser checklist below.

ERROR 81: RECORD NUMBER OUT OF RANGE
If a record number out of range error ocurrs when saving a record, it indicates a corrupted data file. You should use FREL and REINDEX on the file.

ERROR 30: CAN'T READ CONFIGURATION FILE
Configuration file (.FRM) is not a compiled DataFlex configuration or the file is damaged.

ERROR 32: CAN'T OPEN OUTPUT FILE
The disk is full or the file name is improper.

ERRORS 78, 80
Directory is damaged. (above)

L/D: DISK LIMITATIONS

Each file, program and configuration takes up disk space. In particular, if you are using a floppy-disk-based system, you will probably run into disk full conditions. You may run into this at any time whether creating, editing, or compiling any type of file. The error code for a full disk depends on what type of file you are processing. (The error when you are running DataFlex is different from the same error when you are using the EDITOR.) In any case, you will either have to reorganize the files on your disks or get more storage space. Additionally, be aware that the last edit or addition you attempted was probably not saved properly. Note that there are TWO ways to fill up your disk: (1) the storage can be used up; or (2) the directory (number of files) can be filled up. Both of these conditions are treated in the same way; you must remove files from your disk.

The following runtime error codes are returned for disk full.

ERROR 5: DIRECTORY OVERFLOW (DISK FULL)
Directory (number of files) full.

ERROR 6: SEEK PAST END OF DISK (DISK FULL)
Disk space used up.

ERROR 21: WRITE ERROR ON DISK FILE
Disk full while trying to expand index file.

ERROR 32: CAN'T OPEN OUTPUT FILE
Disk directory full while trying to open an output file.

NOTE: The compiler needs disk space on the logged-in drive to store temporary files created during the compile process. Typically, the space needed is approximately the size of the configuration source file. If there is not enough space for the compiler to write its temporary files, Errors 1 or 2 will occur during compilation.

L/M: MEMORY LIMITATIONS

With DataFlex's capacity to relate many data files and have many pages of screens, it is possible for a configuration to simply run out of RAM (random-access memory). There are two factors at work here: the amount of memory available to a program (TPA, for Transient Program Area) and the size of the program or configuration. It is also possible to run out of memory during the compile phase. Compile time minimum memory is different from runtime minimum memory. Minimum compile memory is 1000 bytes. You should find out how much memory your system has available to the program AFTER the operating system is loaded.

A typical computer with a 64-kilobyte Z80 processor can have from 42 to 58K of TPA. In 8-bit systems, a minimum of 52K TPA is required to run DataFlex. On a 16-bit system, you must have a minimum TPA of 100K. This means that if your computer has 128 kilobytes of hardware RAM and your operating system consumes 34K, you do NOT have enough memory to run DataFlex properly. Note that operating systems that control hard disk drives take more memory than comparable floppy-based systems. For example, a 16-bit system with 128K of hardware RAM that runs DataFlex successfully with floppies may not be able to run DataFlex if a hard disk drive is installed unless more RAM memory is also added to handle the larger hard disk driver program. The more memory you have, the larger the configuration you can handle, up to a point. On a 16-bit system, DataFlex will take advantage of a maximum of about 192K of TPA, so the maximum capabilities of DataFlex are likely to be realized with 256K of hardware RAM. Further expansion of RAM would probably yield no benefits to running DataFlex.

When DataFlex is running, it maintains an internal predefined variable called MEMAVAIL, which will return the amount of unused memory available dynamically (the value changes whenever memory usage changes). This amount is what remains after DataFlex and your configuration are loaded. If you run the following test program, it will show you your system's free memory as of the time you run it:

```
/*
SHOWLN MEMAVAIL
SYSTEM
```

The number displayed should NEVER be below 10,000, and if it is above 31,100, your available memory is greater than DataFlex's maximum capability to utilize memory. A result below 10,000 indicates that your computer and operating system as configured do not have sufficient TPA available to run DataFlex.

If you are working with a particularly large configuration that may be close to exceeding available TPA, you should incorporate the procedure shown above at certain points in the configuration so you can keep track of remaining TPA availability. IMPORTANT NOTE: DataFlex requires some memory to be available while running. The amount of memory required at runtime is AT LEAST 2,000 and can be as high as 5,000 if you are using very large data files, so when MEMAVAIL is below 5,000, the possibility of trouble at run time exists, and is a virtual certainty when MEMAVAIL declines below 2,000.

HOW BIG MUST MEMAVAIL BE?

The amount of memory that must be available to run a particular configuration is the greater of: (a) the largest index for the data file(s) used by the configuration; or (b) the largest screen image in the configuration.

A typical screen image is about 2,000 bytes, so the largest data file index is usually the limiting factor. The minimum memory requirement can be estimated with the following formula based on the information returned when an index is created in FILEDEF or REINDEX:

```
(512+KEYSIZE)*(LEVELS+1)
```

Where KEYSIZE is the size in bytes of the key being indexed and LEVELS is the number of ISAM levels required.

Note that LEVELS can be computed by the ISAM module ONLY when the index is created, so take careful note when these numbers are displayed at the end of index creation in FILEDEF, AUTODEF, or REINDEX. The number of levels is based on the **key size** and the **maximum number of records** you define in FILEDEF or AUTODEF. So if you define a file for 30,000 records and it only has 1,000 with data in them, you may be wasting memory! This precaution guards against running out of memory halfway through entering the data to a file, since the maximum memory is used even when the actual data and index files are still small.

WHAT USES MEMORY?

Most of the resources available in DataFlex (or any other language) consume some memory, but there are two areas that typically consume your work area: open data files and screen image windows. Generally if you can do without opening a data file in an application, you will save significant memory. The following table should help you determine what uses memory and therefore where you should look should you have to conserve it. THE FOLLOWING FIGURES ARE IN BYTES.

	8-Bit	16-Bit
COMMANDS		
Each command line	11	11
Indicators	Free - included in commands	
Each string variable	Actual string length + 2	
Each numeric variable	10	10
Each integer variable	Free - pre-allocated	
Each string constant	Actual string length + 1	
Each numeric constant	10	10
Each expression	String length (compacted)	

	8-Bit	16-Bit
SCREEN IMAGES (PER PAGE)		
Overhead (unbuffered)	11	11
Overhead(buffered)	Actual page size	
Each window	Actual window size + 11	
Each subtotal	10	10
Each range check	20	20
Each check string	Actual string length + 1	
DATA FILES (PER OPEN FILE)		
Overhead	120	180
Each field	8	10
Each BATCH index	128	170
Each ON-LINE index	(Number of levels)*512	
Record buffer (actual)	Record length (per FILEDEF)	

If a configuration runs out of memory, it can be divided up into two or more configurations but still run as though one with the CHAIN command (see the section on "Controlling Execution Sequence").

INDICATIONS OF OUT OF MEMORY

ERROR 10 (DataFlex out of memory)
Recursion stack overflow (PASCAL error)
OUT OF MEMORY (compile time)
In some cases, systems may 'lock up' or drop to the operating system

Final note: There are no "secrets" for getting more memory. If there were, they wouldn't be secret; so please don't ask your dealer or distributor for one.

L/C: CONFIGURATION LIMITATIONS

Designing a configuration necessarily involves specifying limits on the application's capacities. Examples of these limits are:

Maximum number of records in a file
Field lengths
Window lengths

Frequently, configuration limitation errors are triggered by errors on the part of the operator, such as entering a number that is too large by mistake. In such cases, the resultant overflow error serves as an automatic form of range verification. If the capacities are not specified thoughtfully, however, they may hamper the purposes which the configuration was designed to serve. If field sizes must be changed (or new fields added, making the record larger) after there is data in the file, the file definition may still be changed without the requirement to rekey the data. First, the data should be written to a new disk file in READ format using the DataFlex QUERY utility, then the file definition changed, and then the data reread back into the restructured file with a configuration built using the DataFlex READ utility.

The following errors are those most frequently caused by configuration limitations:

ERROR 11: NUMBER TOO LARGE FOR FIELD ALLOCATION
This is usually caused by the operator entering too large a number in a data window. Otherwise, a numeric field is just not big enough to hold the required value. You may have to change the file definition to allow the field to hold a larger number.

ERROR 23: INDEX FILE FULL, EXCEEDS DEFINED SIZE
In FILEDEF and AUTODEF you are asked for the maximum number of records that could be in a file. If this number is greatly exceeded, you will get Error 23. To fix this, simply re-enter FILEDEF and change the maximum number of records. You will then have to rebuild all indexes for the file using REINDEX.

CE: CONFIGURATION ERRORS

In any program or configuration, it is possible to use commands which are syntactically correct but which make no sense or cause errors when the progam is run. These are generally cases where the operation of a command is not fully understood or is applied improperly. It may be necessary to put in special "debugging" statements to determine the state of the program while it is running. Still, the most important step is to read the manual carefully and review the sample configurations and programs provided. Due to DataFlex's great flexibility, configuration errors can appear to be other types of errors.

DATA CONSISTENCY

Data consistency errors are those caused by improper operation of the application and its failure to do the checks necessary to guarantee data consistency. An example of this would be the deletion of a customer while there are still transactions for that customer in another data file related to the customer file. In transactional environments it is necessary to prevent the user from deleting such records. One way to do this is to keep a counter of transactions posted to a record and only allow deletion when that counter is zero.

A related error is allowing the user to change the primary key of an active record. It is best to use record number relationships as well as data relationships if you want to allow the user to change key data. Note that in some applications it is normal and acceptable to have "null" relationships, so the application must check for this condition. QUERY can be a valuable tool for inspecting files and diagnosing data problems. Once a problem is uncovered, a check should be put into the application to prevent it from happening again.

Data consistency errors are usually at issue when reports have incorrect totals or data entry screens show the "wrong" records related.

The following errors typically arise from configuration errors:

ERROR 10: +++ OUT OF MEMORY +++
New installation on 16-bit, run SETSCREEn. Otherwise, out of memory, see above.

ERROR 11: NUMBER TOO LARGE FOR FIELD ALLOCATION
Number does not fit into the allocated field; redefine field for a larger value.

ERROR 12: WINDOW NUMBER OUT OF RANGE
Window number out of range. This is usually caused by improper use of window indexing. Recount your windows and/or temporarily print the index value to the screen while the program is running.

ERROR 18: CAN'T OPEN SYSTEM COMMUNICATIONS FILE
DataFlex cannot establish multiuser operating system communications. Check your installation notes for information relating to your particular operating system.

ERROR 19: MULTIUSER TIME OUT (BUSY TOO LONG)
Multiuser time out. The system is in a LOCKED state for too long. Check that none of your programs does a LOCK or REREAD without an UNLOCK. On a very busy system it is possible to get this error occasionally through no fault of the configuration.

ERRORS 25, 41, 42
FIND errors--FIND command was unsuccessful. To determine what the problem is, temporarily print the value of the record buffer to the screen directly before the FIND. Also with error 25, be conscious of the record number, which must be part of the key if included in the index for a FIND EQUAL.

ERROR 28: DUPLICATE RECORDS NOT ALLOWED IN FILE
A SAVE was done on a record having a unique key that matches a key already in the database. If unexplainable, try reindexing the file.

ERROR 31: CONFIGURATION FILE NOT FOUND
Configuration file not found. This is caused by an attempt to load a configuration that does not exist. This can also be caused by an improper menu configuration or an error in a "CHAIN" statement.

ERROR 32: CAN'T OPEN OUTPUT FILE
If the OUTFILE or DIRECT_OUTPUT command cannot create the output file, you will get an error 32. This can be caused by disk full or a bad file name.

ERROR 51: BAD FORMAT IN EXPRESSION (OPERAND)
If the expression evaluator runs across an improper argument when it is expecting an argument, you will get an error 51. Most expression errors are not reported until runtime. Example: (1+*2). If unexplainable, you may have a data consistency error (see above).

ERROR 52: BAD FORMAT OF EXPRESSION (OPERATOR)
If the expression evaluator runs across an improper function (+, -, *, or /) when it is expecting a function, you will get error 52 at runtime. Example: (1=2)

ERROR 71: NO RECORD IN MEMORY TO DELETE
If a DELETE command is issued against an INACTIVE record buffer, the DBMS won't know what record to delete and will report error 71. DELETE must be used with a "FOUND" record.

ERRORS 73, 76, 77: FIELD NUMBER OUT OF RANGE
Field number out of range. This is generated by improper use of field indexing. Carefully check the index values and make sure they are in range. This can also be caused by using an old configuration with a modified file definition.

ERROR 79: FIELD NOT INDEXED, CAN'T FIND BY THIS
Field not indexed on FIND. A FIND command has been issued on an inactive index or a non-indexed field.

ERROR 82: EDITED RECORD NOT SAVED
This error indicates misuse of the database. If an active record is edited (MOVEd to) and **not** SAVEd, you will get an error 82 and **the record will be lost from the index,** so this error should be corrected not only in your application, but in your database as well. After an error 82 has occurred, the file must be reindexed. **Never** MOVE to an active record without a SAVE!

ERROR 83: TYPE CHECK ERROR, ASCII USED AS NUMBER
Type check error. This often arises from use of an old configuration with a database whose file definition has been modified. Update the configuration to take into account the revised file structure.

ERROR 86: RELATED FIELDS ARE NOT THE SAME LENGTH
Related fields MUST be exactly the same length and type. Check your file definitions for all files involved.

ERROR 96: CAN'T OPEN CONFIGURATION/PROGRAM FILE
Configuration/program file not found. This is usually triggered by a CHAIN or RUNPROGRAM command whose target file cannot be found. Review drive assignments of files concerned and make sure they are correct in the command. This is often occurs in MENU because of incomplete drive designation in action specifications.

ERROR 97: MULTIPLE GOSUB'S WITHOUT RETURN
For every GOSUB or use of a KEYPROC, there **must** be a RETURN. If not, after 20 GOSUBs, you will get Error 97. Also if your KEYPROC is called from within a subroutine, you must RETURN to within that subroutine.

SU: SET UP ERRORS

Set up errors refer to problems associated with installation of the DataFlex system or the placement of the required files or the data files on your system. There are some files that are required to be on the default drive and other files that reside on the drive specified by the configuration. Refer to the appendix "File Name and Extension Types" for more information on the types and locations of files. The following files are ABSOLUTELY required on the default (logged in) drive:

AT ALL TIMES:

FILELIST.CFG	(current terminal configuration and location/names of active data files)

AT RUN TIME:

RUN.OVF	(8-bit runtime overlay file)
RUN.00?	(16-bit runtime overlay files)
FLEXERRS.DAT	(Error messages)

AT COMPILE TIME:

COMP.OVF	(8-bit compile overlay file)
FLEX.CFL	(Compiler command file)
*.FD	(File definition file--one for each data file)

WHEN RUNNING SETSCREEN:

TERMLIST.CFG	(List of terminal codes and your serial number)

DataFlex will not run if you have not activated FILELIST.CFG by running SETSCREEn.

INSTALLATION ORDER

The DataFlex distribution disks contain multiple copies of FILELIST.CFG and MENU.DAT. These set up DataFlex properly to run on floppy or hard disk systems depending on the sequence in which the disks are copied. If you don't copy the disks down in the proper (A, B, C) sequence, you will get set up errors.

The following errors commonly arise from improper set up:

ERROR 43: CAN'T OPEN INDEX FILE
The index information is kept in a separate file from the actual data. The indexes have the "ROOT" file name with a ".K?" extension, where "?" is the index number. These files must reside on the same drive as the data file (see Error 75).

ERROR 72: FILE NOT OPEN
If a DBMS file is used before it has been OPENed, you will get error 72. This can also be caused by a file not being on the disk, in which case you should get an error 75 first.

ERROR 74: CAN'T OPEN "FILELIST.CFG"
FILELIST.CFG must be present on the logged in disk drive.

ERROR 75: CAN'T OPEN DATA FILE (.DAT)
The data file name contained in FILELIST.CFG can't be found. The "ROOT NAME" has ".DAT" appended to it for the OPEN. Make sure that the root name in FILEDEF corresponds with the actual name and drive of the data file as shown in your operating system's directory.

ERRORS 92, 96
Configuration/program file not found. This is usually triggered by a CHAIN or RUNPROGRAM command whose target file cannot be found. Review drive assignments of files concerned and make sure they are correct in the command. This can be caused by incorrect installation order.

MU: MULTIUSER ERRORS

If you are running DataFlex multiuser, you should scan this check list before you experience any data related problems.

1) Is multiuser set in SETSCREEn? Option 1 of SETSCREEn must be set to Multiuser. For Option 2, "Pre-allocate data files", refer to the installation instructions provided on loose sheets for your particular computer or operating system.

2) Is the data file set to reread? For a data file to be updated by multiple users, it must be set to re-read, true in FILEDEF.

3) Have you followed each step of the installation instructions? Each multiuser version is a little different. These differences are taken into account in the installation instructions.

4) Is your DataFlex a multiuser version? When DataFlex signs on after the system has been booted, it will show you the multiuser operating system that your copy of DataFlex is to operate on. Make sure this matches (or is compatible with) your operating system. If it signs on "single user" or with an incompatible operating system, you will need to order a different version of DataFlex. Contact your dealer.

5) Is the operating system configured correctly? Most multiuser operating systems require you to set "sysgen" parameters indicating compatibility and the disk volumes enabled to be shared. Make sure you understand all of these options in your operating system and have set them correctly.

The following are indications of improperly configured multiuser systems:

1) Corrupted index files.
2) Operating system error: FILE/DRIVE IN USE or R/O (read only)
3) Only one user can update a data file.
4) Errors 18, 19, 21, 22, 78, 80.

THE DATAFLEX UTILITIES

All of the utility programs in DataFlex except QUERY (FILEDEF, REINDEX, AUTODEF, QUERY, FREL) are intended for operation **only** in single user mode. Make sure that all other users are logged off the computer before using any utilities.

THE DATAFLEX FLEX-KEYS

The DataFlex function keys are active only in the DataFlex Editor and in running configurations. These keys are not active in the DataFlex utilities (FILEDEF, REINDEX, AUTODEF, QUERY, FREL) unless specified otherwise.

CUSTOM ERROR MESSAGES

DataFlex provides the opportunity for the user to create new error messages to be triggered by conditions defined by the configuration. New error messages are defined using error numbers above 100, as described in the explanation of the ERROR command in the manual section on "Controlling Execution Sequence". The facility which provides this capability (a DataFlex configuration and database having the rootname "FLEXERRS") also provides the capability to reword the messages given in the standard errors discussed above. This capability would permit the substitution of messages which might better fit the context of the particular application which you are running, but this caution must be observed in rewording the standard DataFlex errors discussed above: the message substituted will be shown every time that particular error is triggered in any configuration being run on the same disk as the file FLEXERRS.DAT in which the change(s) were made. This may cause an out-of-context error message to appear in a configuration for which the message was not intended.

Finally, the predefined system variable LASTERR and the predefined argument ERR can both be used to provide "chain" messages which will offer helpful prompting and information (either automatic or operator-selected) in the event one of the standard errors above is triggered. ERR is TRUE when any error (standard or custom) is triggered, and it remains TRUE until a CLEARWARNING command is executed. If an error at a particular point in a configuration has a predictable origin, a help message or execution branch can be provided based on the status of the ERR indicator. If, on the other hand, the triggering of one or more standard errors has consistent implications throughout a particular configuration, IF commands based on the value of LASTERR (IF LASTERR EQ some particular error number), can cause the appropriate prompting or execution branching to assist in efficient recovery from the error.

THIS PAGE INTENTIONALLY LEFT BLANK

GLOSSARY

APPLICATION
A task, or set of tasks, to be performed by a computer, typically through configurations (or programs) and a language, such as DataFlex. Database management is not an application in itself, but Accounts Receivable billing may properly be described as one database management application.

AUTODEF.COM
The program in DataFlex for quick and easy file creation. AUTODEF will set up the file definition as well as build a data entry configuration file which can either be custom-edited, or compiled as-is for execution.

B+ or B* TREE
A fast and efficient method of searching a data file for a key value used by DataFlex.

BATCH UPDATING
The technique of updating files from a transaction file as opposed to on-line or real-time updates at the time of operator input.

BIT
The smallest unit of information used by a computer. A sub-unit of a byte. Eight bits = 1 byte and four bits = 1 nibble.

BUFFER, RECORD
An area of memory (RAM) that is devoted to the current record. REMEMBER, data or information is: (first) in the file; (second) in the buffer; then (last) on the screen or in the window.

BYTE
Eight bits (usually representing a character). The smallest unit of information which can be handled by a DBMS.

CHAIN or CHAINING
Moving from one program or program module to another without operator intervention.

COMMAND ACTION
Those commands or controls used/invoked by the user or operator during data entry. The soft keystroke commands set with the program SETSCREE.COM.

COMMAND GROUP
The sections of an enter or report configuration which are invoked when the operator performs a corresponding command action. The operator key strokes are set with the SETSCREEN program.

COMMAND LINE
An expression string executable by the operating system (DOS), typically from the CP/M prompt A>. Usually these are issued by MENU in a completed DataFlex application. In DataFlex 2.0 these can be executed from configuration command lines.

CON:
Customary designation of the output port of the computer which communicates with the system's console, or screen.

CONFIGURATION
A logical asssembly of DataFlex commands and values which, upon execution, cause DataFlex to perform a task (see Application above). Configurations are to DataFlex what programs are to traditional computer languages.

CONFIGURATOR
Person who writes DataFlex configurations.

DAC
Data Access Corporation, the publisher of DataFlex.

DATA
The real world information contained in the files of the Database. Usually the smallest unit is a field of a record in a file. Data is usually stored as one of two types, formatted and unformatted. DBMS's deal in formatted data, whereas word processors (for example) deal in unformatted data.

DATA DICTIONARY
A catalog of all data elements in a database giving their definition, format, and source. In DataFlex, the File Definitions collectively.

DATA STRUCTURE DIAGRAM (DSD)
The logical structure of the database design showing the relationships of the files and their data.

DATA TYPES
In DataFlex, four data types are recognized. They are alphanumeric, numeric, integer and date.

DATABASE
A computerized collection of stored operational data that can serve the needs of multiple users or applications. The collective files that make up an application in DataFlex. In this manual, the two forms are referred to with the same term. Some authors mean other things and make a distinction between them.

DBMS
Data Base Management System. Traditionally two classes: (1) Stand alone; and (2) Higher level language sub-set.

EDITOR
A program which handles unformatted data which is typically a document or source program. A text file handler. A word processor incorporates a powerful EDITOR.

ELEMENT
The smallest discrete unit of data handled by a data management system. In a mailing list, for example, a data element might be first name, or ZIP code. Also referred to as "field".

ERGONOMICS
The study or science of human factors in the design of devices for human use such as chairs and computer moni tors.

EXTENSION
That part of the directory file name to the right of the dot (up to 3 characters), often used to classify files by type or function. See ROOTNAME and FILE below. In DataFlex, extensions are used for specific meanings:

.DAT = Data file	.FRM = Source code for data entry
.Kn = Index file (n=#)	.TAG = Database field names
.FD = File definition file	.CFG = System information file
.FLX = Compiled program	.RPT = Source code for reports

FIELD
A sub-unit of a database record. The information unit. Also called "data element".

FILE
The basic unit of information stored on a disk drive which can be found by a computer's operating system. Each file on a disk drive has a unique name (see ROOTNAME and EXTENSION) which is found both by the computer and by users in the operating system's directory of files on a drive. The source code for a configuration is typically contained in a FILE, as its compiled version is contained in another FILE. Database data is contained in another FILE, and index data for that FILE is contained in yet another FILE.

INDEX
A table containing ordered information on records or data elements in a database and their locations.

INDEX FILE
In DataFlex, a B+ ISAM file containing information about a data file in order by a key of selected data elements within the data file.

ISAM
Indexed Sequential Access Method. An access method associated with a sequential data file whose contents are ordered by primary key values.

JULIAN DATE
A numeric conversion of dates used internally by DataFlex to facilitate rapid, accurate calculations of dates and calendar time intervals. It is a form of the number that represents the number of days from the year Zero to whatever date is being processed. The original method was developed in the 16th Century by Joseph Scaliger, and contrary to popular assumption, its name derives from Scaliger's father, Julius, and not from Julius Caesar or another calendar named after, and decreed by, him.

KEY
The field or portions of a record used to uniquely identify that records in a database.

LOGGED-IN DEVICE
The default logical device that the operating system will load programs and data from (typically A:).

LOGICAL DEVICE
A peripheral data input, output, or storage device addressed by the convention of one or three letters and a colon. Common examples:

B:	The "B" floppy or hard disk drive
CON:	The console, or screen
LST:	The listing device, or printer
RDR:	The punched tape or card reader
PUN:	The paper tape or card punch

LST:
Customary designation of the output port of the computer which communicates with the system's LIST drive, or printer.

MENU / MENUDEF.FLX
MENUDEF in DataFlex is an interactive program which creates menus, issues query prompts to the operator, and issues operating system and DataFlex commands. MENUDEF is itself entirely menu-driven.

O/S (OPERATING SYSTEM)
The disk operating system (DOS).

ONLINE UPDATE
Updating information in the files/database at the time of operator input.

QUERY
The function of searching for and retrieving information from a database according to operator-specified criteria based on logical operators such as less than or equal to. In DataFlex, a user friendly interactive program (QUERY.COM) which can be used to generate reports and report configurations.

RECORD
A sub-unit of a data file, in turn divided into fields.

RELATE / RELATIONSHIPS
A logical the records of one data file to the records in another by having one or more pairs of fields contain IDENTICAL data in both files. In DataFlex relations are defined using the FILEDEF program. Refer to the File Definition section of this manual.

ROOTNAME
That part of the directory file name to the left of the dot (up to 8 characters), typically entirely under the user's discretion to use in identifying related "families" of files. In DataFlex, a common rootname identifies the various data files which together constitute a single database. See EXTENSION and FILE above.

SELECTION
The logical operators for selectively retrieving information from a data base file. Used in combination, they can perform the function of ranging, (i.e. greater than x & less than y).

SETSCREEN
An interactive installation procedure required when installing DataFlex, by means of which DataFlex is provided with the specific screen control codes for the computer or terminal on which DataFlex is to be run.

TPA
Transient Program Area. That area of the computer's memory which is available to hold programs and data after the DOS or disk operating system is loaded and running. On 8-bit CP/M systems, DataFlex requires a minimum of 52K of TPA, and 100K for 16-bit systems.

USER / OPERATOR
The final user or operator of a designed system. Typically the data entry clerk. A non-technical computer user.

WINDOW
The position in which information (data) is displayed on the screen during ENTER or in a report.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF ASCII CODES

CTRL	CHR	DECIMAL	HEX	CHR	DECIMAL	HEX	CHR	DECIMAL	HEX
@	NUL	000	00	+	043	2B	V	086	56
A	SOH	001	01	,	044	2C	W	087	57
B	STX	002	02	-	045	2D	X	088	58
C	ETX	003	03	.	046	2E	Y	089	59
D	EOT	004	04	/	047	2F	Z	090	5A
E	ENQ	005	05	0	048	30	[	091	5B
F	ACK	006	06	1	049	31	\	092	5C
G	BEL	007	07	2	050	32	]	093	5D
H	BS	008	08	3	051	33	^	094	5E
I	HT	009	09	4	052	34	_	095	5F
J	LF	010	0A	5	053	35	`	096	60
K	VT	011	0B	6	054	36	a	097	61
L	FF	012	0C	7	055	37	b	098	62
M	CR	013	0D	8	056	38	c	099	63
N	SO	014	0E	9	057	39	d	100	64
O	SI	015	0F	:	058	3A	e	101	65
P	DLE	016	10	;	059	3B	f	102	66
Q	DC1	017	11	<	060	3C	g	103	67
R	DC2	018	12	=	061	3D	h	104	68
S	DC3	019	13	>	062	3E	i	105	69
T	DC4	020	14	?	063	3F	j	106	6A
U	NAK	021	15	@	064	40	k	107	6B
V	SYN	022	16	A	065	41	l	108	6C
W	ETB	023	17	B	066	42	m	109	6D
X	CAN	024	18	C	067	43	n	110	6E
Y	EM	025	19	D	068	44	o	111	6F
Z	SUB	026	1A	E	069	45	p	112	70
[	ESC	027	1B	F	070	46	q	113	71
\	FS	028	1C	G	071	47	r	114	72
]	GS	029	1D	H	072	48	s	115	73
^	RS	030	1E	I	073	49	t	116	74
_	US	031	1F	J	074	4A	u	117	75
	SP	032	20	K	075	4B	v	118	76
	!	033	21	L	076	4C	w	119	77
	"	034	22	M	077	4D	x	120	78
	#	035	23	N	078	4E	y	121	79
	$	036	24	O	079	4F	z	122	7A
	%	037	25	P	080	50	{	123	7B
	&	038	26	Q	081	51	|	124	7C
	'	039	27	R	122	52	}	125	7D
	(	040	28	S	123	53	~	126	7E
	)	041	29	T	124	54	DEL	127	7F
	*	042	2A	U	125	55			

THIS PAGE INTENTIONALLY LEFT BLANK

INDEX

OPEN Command

An OPEN command makes all database files associated with a file "df_filename" (data and key files) available for DataFlex operations. (See the FILEDEF documentation for information on df_filename). Specifically, when a file is OPENed, the referenced data file, and all associated key index files are opened, buffer space is automatically allocated in memory, and upon completion of the OPEN operation, all files are available for on-line processing.

The OPEN command(s) should be at the beginning of the configuration and **must be executed before the files associated with the df_filename can be addressed by other commands.**

FORMAT:

> **OPEN file_name_arg {index.n}**
> Where OPEN is the command, "file_name_arg" is the "SHORT NAME" (created in FILEDEF) of the file to be OPENed, and n represents the number of an index that may optionally be loaded into a memory "buffer" to speed sequential access to records in a file.

EXAMPLE:

```
OPEN inventory
OPEN personnel index.2
```

In the second example above, the "index.2" option after the df_filename "personnel" elects to load the second index for that database file into a memory buffer.

The optional index buffering only provides a speed improvement when the access to records is sequential, as in a report. Random operations, such as transaction entry to non-sequential records, will not benefit from buffering.

The usefulness of the index buffering option is, as with most other options, subject to tradeoffs. The number of the index is established in the index definitions as previously created in the FILEDEF function. If the option is selected, record locations required by FIND and other accessing operations are read directly from the computer's active memory (RAM) rather than via a separate (slower) disk access and file-read operation.

The tradeoff when using record buffering is the sacrifice of a portion of the computer's RAM that is consumed by this process. The overall size of your configuration, number of files opened, RAM capacity of your computer, and so on, determines how much RAM is available over and above what is required to carry out the application's basic functions, and will dictate whether the index-buffering option is worthwhile, or even possible.

PLEASE READ THIS BEFORE BREAKING THE SEAL ON THE DISKETTE ENVELOPE

DataFlex Registration: What, Why and How

Registration is an easy, simple procedure which permanently marks your copy of DataFlex as exclusively yours or your company's. It also documents that you have rights to use DataFlex. When you start the enclosed DataFlex software, it will display the "TEMPORARY REGISTRATION" legend shown on the reverse side of this document. After registration, the sign-on will display your chosen name. This is also illustrated on the reverse side of this document.

First the attached "Limited Use Software License Agreement" must be signed and returned to Data Access Corporation. We cannot register your license without this Agreement. The Agreement describes what you may and may not do with your DataFlex software and its documentation.

If the terms of the agreement are unacceptable to you, DO NOT BREAK THE SEAL on the envelope containing the disks. Take the envelope, documentation, and these papers back to your dealer and request a refund.

If, as we hope, all is agreeable, sign and date the Agreement. Turn it over and referring to the instructions on the back of this page, carefully fill out the "Registration Request". After you have done this, you may want to make and keep a copy of the Registration Request and License Agreement for your records. When everything is complete, place only the Registration Request/License Agreement in the provided return envelope and mail it back to us. We will get back to you with your registration code.

If both the Registration Request and the License Agreement are not completely filled out, they will be returned to you without processing your registration request.

With these papers, you will also find a sheet of keycap appliques. There is an applique for each single-key command (Flex-Key) in DataFlex as well as some others. After you have installed DataFlex, you will find these appliques handy for marking the keys you have selected (with the SETSCREE program) for each of these functions.

Please turn this page over for registration instructions.

How to fill out the Registration Request

The Registration Request on the next page is the way you tell us in what name you want your copy of DataFlex registered. The registered name should be the owner (a company or individual) of the license to use DataFlex. Capitalization and punctuation of the name are up to you. If these factors are important to you, be very careful. A registration can not be changed once it is done.

Please be careful to also enter the correct address. We have to mail your registration code back to you. In addition, your registered address is used to mail product update notices, technical bulletins and similar material to you.

After you have run SETSCREE to install DataFlex on your system, DataFlex will sign on with a screen similar to the one shown below.

```
DEFAULT DRIVE=A:
OS: (name of your operating system)

This program licensed to:  TEMPORARY REGISTRATION
           Serial Number:  ####  (your serial number)

2.1/ DataFlex
Copyright (C) 1984, Data Access Corporation, Miami Florida
All rights reserved.
```

If your sign-on displays *DEMO VERSION*, company name registration does not apply and you may ignore the registration procedure. Your dealer will be happy to assist you in acquiring a full-function DataFlex license.

When we receive your filled out Registration Request, we will use the exact name you entered on the top line to generate a unique access code. This will enable you to replace the DataFlex TEMPORARY REGISTRATION sign-on with the name from your Registration Request. You will receive complete instructions for installing your registration name with your code number. If you had requested registration for "ACME Manufacturing" and your operating system was CP/M and your serial number was 25897, after installation your sign-on would look like the following.

```
DEFAULT DRIVE=A:
OS: CP/M SINGLE USER

This program licensed to:  ACME Manufacturing
           Serial Number:  25897

2.1/ DataFlex
Copyright (C) 1984, Data Access Corporation, Miami Florida
All rights reserved.
```

DataFlex Registration Request

ENTER THE EXACT "REGISTRATION NAME" IN THE BOXES BELOW:

|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

Place only one character (letter, space, comma, etc.) in each box. Any character that you can type on your keyboard is allowed. Do not exceed the number of boxes provided (leave excess boxes blank). **Upper and lower case letters may be used, but each letter must be distinctly one or the other. For clarity, please place an "X" beneath any letter that is to be lower case.**

The DataFlex serial number can be found on your DataFlex master diskettes. Your license type is Full Development if you have both a Runtime and a Development disk. It is Runtime if you have only the Runtime disk.

Please enter your DataFlex serial number here: ____________________

License Type: Full Development License ____ Runtime License ____
NOTE: Demo licenses do not receive Access Codes - contact your dealer.

We will enter your access code here: ____________________
(leave blank)

Please complete the following information about your system:
Computer Manufacturer ____________________
Model ________________ Serial # __________

Operating System __________ Rev. ______ 8-Bit or 16-Bit ____

DataFlex Purchased From ____________________

Your Name ____________________

Your Company ____________________

Address1 ____________________

Address2 ____________________

City __________ State/Prov ________ Zip/Postal Code _____

Country ____________ Phone ________ Telex ____

My computer is: Single User ____ Multi-user ____ # Terminals _____
Floppy Disk ____ Hard Disk ____
For what applications do you plan to use DataFlex?

____

Are you using DataFlex in your own business?	(Y/N) ____
Developing applications for another business?	(Y/N) ____
Do you use an outside consultant/programmer?	(Y/N) ____

PLEASE RETURN THE COMPLETED FORM TO DATA ACCESS CORPORATION. A COPY OF IT, WITH YOUR ACCESS CODE, WILL BE RETURNED TO YOU UPON VERIFICATION OF YOUR LICENSE. PLEASE BE CERTAIN TO COMPLETE THE AGREEMENT ON THE OTHER SIDE. REMEMBER, DEMO LICENSES DO NOT RECEIVE AN ACCESS CODE.

Limited Use Software License Agreement

Data Access Corporation (DAC), a Florida, USA corporation, does hereby grant to LICENSEE a personal, non-transferable, and non-exclusive license to use the following computer software programs (singularly or collectively referred to as "software") under the following terms and conditions.

Software: DataFlex 2.1 DataFlex Serial Number: ____________

Solely for use on Computer ______________________________, Serial # ____________

(A separate Limited Use Software License Agreement and DataFlex Registration Request must be executed for each software product licensed.)

1. The software is furnished to LICENSEE for use on the single computer specified above and may be used only on that computer. LICENSEE and LICENSEE's employees and agents will protect the confidentiality of the Software and will not distribute or otherwise make available the Software or documentation, or any portion thereof, in any form to any third party. Title to and ownership of the Software and the documentation shall at all times remain in DAC.

2. No warranties with respect to the Software are made by DAC. DAC DOES NOT WARRANT THAT THE SOFTWARE WILL MEET SPECIFIC REQUIREMENTS OF LICENSEE. DAC SPECIFICALLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. DAC SPECIFICALLY DENIES ANY LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OF THE SOFTWARE.

3. This agreement, the license granted hereunder, and the Software, may not be assigned or transferred by the LICENSEE without the prior written consent of DAC except that LICENSEE may change the installation location upon notice to DAC and dealer. No right to reproduce or copy any documentation, in whole or in part, is granted to LICENSEE.

4. LICENSEE acknowledges that it is receiving only a limited license to use the Software and related documentation and that DAC retains title to all Software and documentation. LICENSEE acknowledges that DAC has a valuable proprietary interest in the Software and documentation and that unauthorized disclosure, transfer, or copying of the Software is a violation of Federal law.

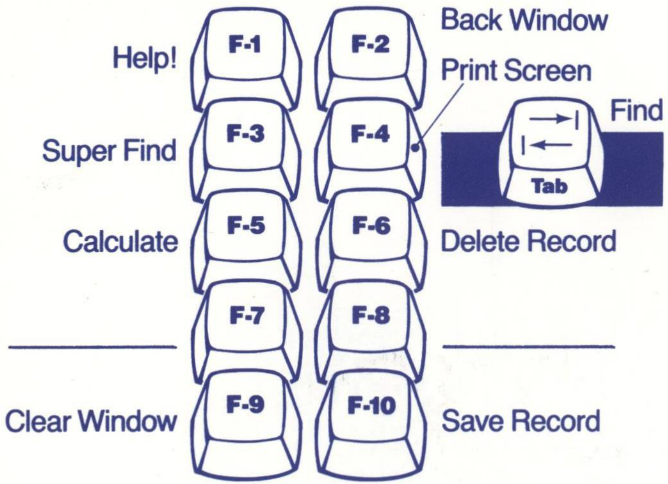
LICENSEE Name ____

Authorized Signature ______________________________ Date ____________

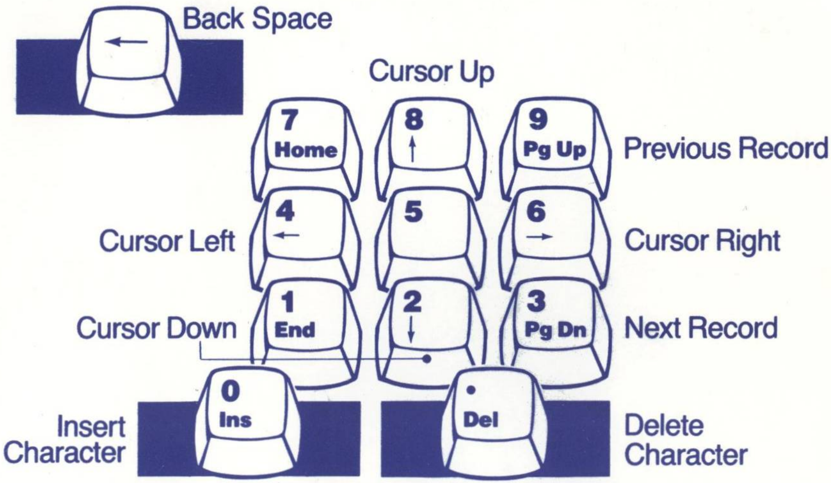
Signer's typed or printed name ______________________________________

DataFlex® Flex-Keys™ For The IBM PC

Function Pad Flex-Keys™



Cursor Pad Flex-Keys™


Back Space
Cursor Up
7 Home
8
9 Pg Up
Previous Record
Cursor Left
4
5
6
Cursor Right
Cursor Down
1 End
2
3 Pg Dn
Next Record
Insert Character
0 Ins
. Del
Delete Character